Aalto-yliopisto
Aalto-universitetet
Aalto University

Nordic Master Programme in Environmental Engineering

Identification of Conceptual Models for Sewer Flow

Ha Viet Nguyen

Master's Thesis
2020

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Ha Viet Nguyen

Title of thesis Identification of Conceptual Models for Sewer Flow

Programme Nordic Master Programme in Environmental Engineering

Major Environmental Engineering

Thesis supervisor Senior Lecturer Teemu Kokkonen

Thesis advisor(s) Associate Professor Roland Löwe, Associate Professor Morten Borup

Date 01.11.2020 **Number of pages** 60 + 55 **Language** English

Abstract

Climate change will cause more extreme events to happen in the future, along with which is the higher frequency of rain. Besides, the growth of population and urbanization also results in more burden to the urban drainage systems. In order to understand the response of a drainage system, physically-based distributed models (PDM) such as Mike Urban+ are often used. However, these models require intensive computational power, which makes them not feasible for scenario analyses or real time control. Conceptual models are preferred since they can produce fast simulation results. Nevertheless, conceptual models' parameters are concepts that cannot be directly derived from reality, but through calibration or establishment of empirical links with the physical systems.

This study proposes a method to derive parameters of the conceptual models based on the drainage systems' own physical characteristics using the storage-outflow (SQ) relationship. Two conceptual models are built in Python script. The first one optimizes and computes the SQ relationship for transport stretches of various lengths and slopes. The second model uses the obtained SQ relationship to simulate two virtual cities' discharges, each city is divided into 4 different levels of lumping.

The conceptual model for transport stretch performs accurately with low errors. However, discharge hysteresis in long and flat pipes cannot be captured precisely. In comparison to using Mike Urban+, the conceptual models for simulating city discharges are up to 70 times faster with very high accuracy. The NSEs of different lumping levels are around 0.94 while the peak errors both in magnitude and time are small. It is recommended that the cities should be divided into compartments with the areas of around 7–15 ha for high quality results and low computational time. The quality of conceptual models does not depend on the shape of the city. Results show that the conceptual models in this study are capable of simulating outflow independently of any other data than the physical characteristics of the pipes.

Further work still needs to be conducted to evaluate the stability of the conceptual models and increase its accuracy. However, this study's conceptual models show great potential for fast and accurate discharge simulation in an urban drainage system without the need of using any other PDM.

Keywords conceptual model, urban drainage, storage-discharge relationship, physical characteristics, transport stretch, compartments.

Acknowledgements

This study was conducted as a master's thesis for the Technical University of Denmark (DTU), Department of Environmental Engineering and Aalto University, Department of Built Environment. The period of this thesis is 6 months, from June 1^{st} 2020 to November 1^{st} 2020. The thesis is worth 30 ECTS points. This thesis was supervised by Associate Professor Roland Löwe, Associate Professor Morten Borup, Professor Karsten Arnbjerg-Nielsen from DTU and Senior University Lecturer Teemu Kokkonen from Aalto University.

I would like to express my great appreciations to all my supervisors for their invaluable inputs and guidance throughout the project. Especially, I would like to thank Roland Löwe for his patience in helping me with starting up the project that I did not have so much knowledge on, and with the initial codes to give me ideas of how to develop my models.

Furthermore, I would like to thank my family and friends who have been supporting me immensely both physically and mentally during this period. My family has been the strongest base for me, encouraging me through the difficult time of Covid-19, while fully helping me between moving to Vietnam and back to Denmark.

Additionally, I want to deeply thank the person behind all my success, for without this person's sacrifice, I would never be able to achieve what I had today. Finally, I would like to thank my partner for the tremendous amount of help and patience given to me when walking me through many new concepts.

Acronyms

CDS Chicago Design Storm.

CM conceptual model.

IDF curve intensity-duration-frequency curve.

MU+ Mike Urban plus.

NSE Nash-Sutcliff efficiency.

PDIFF difference in peak.

PDM physically-based distributed model.

PEP percentage error in peak.

PTDIFF difference in peak time.

RMSE root mean square error.

SQ storage and outflow.

SVE de Saint-Venant equations.

Contents

1 Introduction

It is a well-known fact that climate change is and will affect the intensity of precipitation in the future, likely causing more extreme storm events with higher frequency in most parts of the world. Under such circumstance, the risk of severe floods as a consequence of high intensity storms is projected to increase significantly, affecting human's lives as well as the stability of the ecosystem (Allen et al. 2012; Hirabayashi et al. 2013). Along with that, the growth in population and the increasing rate of urbanization, with more than half of the population living in cities now, pose huge stress on the urban water management system (UN 2018). Urbanization leads to more houses, buildings, paved roads and other impervious structures, thus reducing the infiltration of water and altering the route of stormwater. For these reasons, cities have an increasing amount of wastewater and stormwater and are the most vulnerable to flooding during heavy rain (García et al. 2015). Urban drainage systems are also under great stress, having to operate with capacities higher than those they were designed for (Ven Chow 1988).

Modelling is a powerful tool to understand the response of a drainage system during a storm. By using models, it is possible to implement real time control or to simulate various hydrological scenarios for planning. Urban drainage system's behavior is often modelled using physically-based distributed models (PDMs) such as Mike Urban+ (MU+), which offer detailed information about water flow and water level in both space and time. However, due to a large amount of calculations, PDMs are often computationally intensive, meaning that it can take a long time to produce results. Moreover, it is not always necessary to compute flow in every single pipe, while a simple model taking some input and producing some simple output at a certain point of the network can be sufficiently accurate. In this case, the use of conceptual models (CMs) is more favored. A conceptual model is a type of model that relies on the concepts of hydrology to simulate flow in a lumped way, i.e. only at several predefined places, using mathematical equations (Kroll et al. 2017; García et al. 2015; Wolfs, Villazon, and Willems 2013). These concepts are represented by parameters. Nevertheless, these parameters cannot be directly derived from reality, but often obtained from either calibration against observed or PDMs' data, or the establishment of an empirical link to the physical system's characteristics. The latter method is especially crucial in the context of planning and design, where many scenarios need to be examined, while no calibration beforehand is possible. Therefore, there needs to be a simple conceptual model whose parameters do not involve any calibration but can be derived directly from the drainage system's physical characteristics.

1.1 Goal and Scope

The main aim of this study is to build conceptual models using custom coded scripts to reproduce the discharge simulated by Mike Urban+. By doing so, we can have a fast and efficient model that can replace the computationally expensive physical models.

For the CM to work, it is important to calibrate the model parameters and decide the level of lumping of the network, i.e. the number of compartments. The CM will rely on the storage-discharge relationship in an urban drainage system. This relationship (i.e. the CM's parameter) is derived empirically in this study, starting from the behaviour of pipes having certain physical characteristics. With this method, it is possible to then simulate any network without the need to calibrate the model against PDM or observed data. After the model parameters are found, the effect of simplification is also studied by examining various sizes and numbers of compartments for two virtual cities of different shapes.

The research question of this study is:

How to build a fast and reliable conceptual model to simulate water discharge from a drainage system based on its physical characteristics?

To answer this, sub-questions are drawn:

- How can a transport stretch be conceptualized based on its length and slope?
- When dividing the city into different compartments, the conceptual model needs to use as input, parameters such as the lengths and diameters to characterize such compartment. How should parameters such as length and diameter be calculated to describe a lumped city compartment, based on the lengths and diameters of each pipe in the compartment?
- When lumping, how many parts (i.e. compartments) should a city be divided into, in order to accurately simulate the discharge out of it?
- How does the conceptual model perform with different city shapes?

The conceptual model in this study will focus only on simulating discharge inside the drainage system of virtual cities. Dry weather flow and combined sewer overflow (CSO) will not be considered.

2 Literature Review

This chapter provides background knowledge on the subjects related to this study, including:

- Model types
- Storage-outflow models

2.1 Model Types

A model is always a simplification of the real world, made to predict or understand the behavior of a system. A good model should be capable of simulating results closely to reality while being simple and providing good results with a small amount of input (Devia, Ganasri, and Dwarakish 2015). Regarding the simulation of various components inside the water network, there have been numerous approaches and models, which can fall into different classification systems. However, this study focuses only on the two most popular types of models, which are the physically-based distributed model (PDM) and the conceptual model (CM). The main difference between these two is the level of detail as well as how they are structured physically.

2.1.1 Physically-based Distributed Model

The PDMs are also referred to as models for hydraulic routing or distributed flow routing (Ledergerber et al. 2019). There are numerous PDMs worldwide, a few examples could be: MIKE from DHI, SWAT (Soil and Water Assessment Tool), SWMM (Storm Water Management Model).

This type of model is capable of computing flow characteristics (velocity, piezometric level, etc.), meaning that they can produce various outputs at once, both spatially and temporally by using the de Saint-Venant equations (SVE). These are 2 equations that coupled together describe the conservation of mass and energy that characterizes the flow into kinematic wave, diffusive wave, and dynamic wave. Depending on the aim of the simulation, the energy conservation equation can be solved either fully or partially (i.e some terms can be neglected), with the fully solved equation resulting in the highest accuracy (García et al. 2015). An analytical solution for the SVE is not available. However, there exist various numerical solution schemes (Wagener, Gupta, and Wheater 2008).

While being able to simulate flow accurately with high resolution time-wise and space-wise, there are still several drawbacks of PDMs. The biggest problem is that they require long computation time due to their high level of detail. Along with it, numerical instability is another issue. The amount of data needed as input to this model is relatively large and not always available. (Butler et al. 2018).

2.1.1.1 Mike Urban+

Mike Urban+ (MU+) is an urban water modeling software with integrated GIS that is able of simulating discharge and water levels of both the collection system and water distribution system. The rainfall-runoff simulation is included inside the collection system module, along with other types of simulation such as pipe flow, pollution transport, biological processes, and control. The rainfall-runoff module consists of models like time-area, kinematic wave, linear reservoir, unit hydrograph surface runoff, and rainfall-dependent infiltration. The type of model used in this study is time-area, which computes the runoff based on each catchment's time of concentration. (DHI 2020).

The collection system module uses Mike 1D engine. This is a hydrodynamic engine capable of simulating flow (assumed to be uniform in velocity) of any type of network structure by using the SVE, including flow from the free surface, open channel, pressurized, or from weirs and pumps, etc. The high order fully dynamic wave equation makes it possible to compute fast transients, tidal flows, rapidly changing backwater effects and flood waves, and steep channels (DHI 2020). The numerical solution for SVE used by Mike 1D is the implicit finite-difference scheme, which solves the current and next state of the model and can be considered the most accurate and stable solution (Wolfs, Villazon, and Willems 2013).

2.1.2 Conceptual Model

The conceptual model (CM), also known as the hydrological model, relies on using conceptual relationships (such as storage and discharge) to calculate the flow. It means that the conservation of energy is disregarded and only the law of mass conservation is respected. A CM often lumps hydrological processes spatially and only calculates flow rate based on the time at a certain location. As a result, flow calculation cannot be made for intermediate points and it does not consider the water level or flow velocity. Another principle of CMs is that downstream flow does not influence the flow of upstream pipes. For these reasons, a CM cannot describe phenomena such as the backwater effect or pressurized flow directly. However, a significant advantage of CM is that it enables fast simulation time where a high level of precision and detail is not necessary (Knight and Shamseldin 2006; Meert et al. 2014). The parameters used for CM often require calibration by optimization based on a constructed PDM to utilize all the detailed knowledge offered by PDM (Gomez 2011).

The important aspects of a CM are how the compartments are discretized and how the flow is calculated. Compartments are defined as delineated areas within the catchment, where water is computed based on the inflow from upstream and outflow to downstream compartments. Figure 2.1 shows the difference between modeling an entire pipe network and modeling compartments. In the latter case, several pipe branches are lumped together to make a compartment, and the modeling process will only consider the compartments, not every single pipe anymore.

For the delineation of the compartment, both Kroll et al. (2017) and Thrysøe, Arnbjerg-Nielsen, and Morten Borup (2019) agreed that the boundary should be drawn where there are flow altering structures, especially where there are throttle pipes. Subdividing compartments is dependent also on the intention of the modeler, but in general, the more compartments there are, the more accurate the model becomes (Thrysøe, Arnbjerg-Nielsen, and Morten Borup 2019; Ledergerber et al. 2019). However, this results in a more complicated model with more parameters to calibrate.

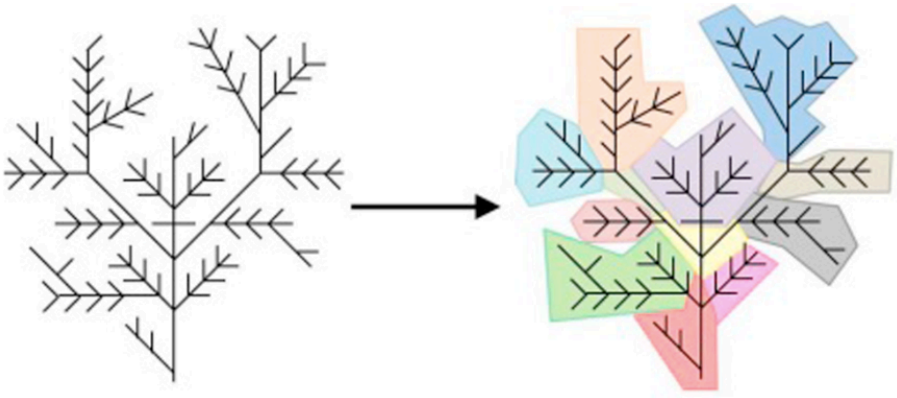

Figure 2.1: Example of compartment delineation (Thrysøe, Arnbjerg-Nielsen, and Morten Borup 2019)

Another issue to address when building a conceptual model is how to describe the flow of water throughout the catchment at different types of structures. Advanced models exist, such as KOSIM or CITY DRAIN. KOSIM is a conceptual model software that derives runoff from rainfall by using 3 linear reservoirs in series with the storage constant of each reservoir being $1/4$ of the catchment's concentration time. CSO is modeled in KOSIM by using an inflow-discharge relationship, while the rest of the water is transferred to an outflow weir (Meert et al. 2014). KOSIM can also be applied to other PDM such as WEST for the integration of water quantity and quality control (Solvi et al. 2005). CITY DRAIN is a more complex model using also the linear reservoirs in series concept. However, this model allows either inflow only into the most upper sub-reservoir, or the distribution of inflow to all sub-reservoirs. CSO is simulated using the discrete mass balance equation (Achleitner, Möderl, and Rauch 2007).

A rather popular and simple approach is to use a conceptual storage-discharge relationship. The detail of this relationship will be described in section 2.2. Thrysøe, Arnbjerg-Nielsen, and Morten Borup (2019) used a piecewise linear relationship and apply it to describe all the flows out of a compartment, including surface spill, weir discharge, or combined sewer overflow. The relationship was achieved by taking the resulting volume and discharge caused by certain rain data from a PDM and then fitting piecewise linear lines to describe the relationship between storage

and discharge. Kroll et al. (2017) used the concept of recursively calculated linear reservoir cascade to characterize flow routing within the catchment. For the flow in throttle pipes, he uses the Manning equation, given that the water level at the upstream and downstream of the pipe is known. The water level is computed by deriving a lookup table between the storage volume and water level based on the results of a hydrodynamic model.

As can be seen, both models mentioned above are still highly dependent on other PDMs or measured data for calibration or their parameters to obtain the storage-outflow function. It means that without already known results, conceptual models like these cannot be built. In this thesis, this issue will be addressed by establishing a storage-discharge relationship that is only based on the characteristics of a pipe.

2.2 Storage Outflow (SQ) models

In hydrology, the term "flow routing" describes a method of obtaining the flow hydrograph of a downstream point given that the hydrographs of some upstream points are known. In other words, it can be understood as a procedure to follow water flow in a system when the input to the system is available. A lumped flow routing, which is the case of consideration in this study, is the method of calculating flow through the system varying through time only, while the aspect of space is disregarded. (Ven Chow 1988)

The central component of flow routing is the use of the conceptual storage principle. This principle works on the basis of the continuity equation that links the inflow and outflow of a system to storage (Equation 2.1). Most importantly, this principle can be applied not only to flow routing but can be flexibly used to model rainfall-runoff processes.

$$\frac{dS(t)}{dt} = I(t) - Q(t) \tag{2.1}$$

Where $I(t)$ is the input (inflow) to the system at time t, $Q(t)$ is the output (outflow) from the system at time t and $S(t)$ describes the storage of the system at time t. Equation 2.1 indicates that the storage of a system is the difference between the inflow and the outflow. When the storage is known, the discharge hydrograph $Q(t)$ can be obtained, making storage the most important parameter in conceptualizing a drainage system (Vaes and Berlamon 1997).

However, with the known inflow $I(t)$, it is still not possible to determine the outflow hydrograph $Q(t)$, as both Q and S are unknown. For that reason, there needs to be a transfer function, or a storage function, to connect these three parameters together to couple with the continuity equation. The general form of this function can be written as in Equation 2.2.

$$S = f(I, Q) \tag{2.2}$$

Equation 2.2 describes the storage function as dependent on both inflow and outflow. Nevertheless, this study only considers the cases where storage is related to only outflow of the system. It means that in this case, the relationship with inflow is only considered with Equation 2.1, not the following Equation 2.3.

$$S = f(Q) \tag{2.3}$$

Equation 2.3 states that storage can be either a linear or nonlinear function of the outflow. This relationship will be referred to as the SQ relationship, or SQ function.

2.2.1 Linear Reservoir Model

A special case of this SQ function is the linear reservoir, where the relationship between storage and discharge is a single straight line, as can be seen from the following equation:

$$S = c\,Q \tag{2.4}$$

In Equation 2.4, c is the empirically-derived storage coefficient that has the dimension of time (seconds), S is the storage (m^3) and Q is the outflow (m^3/s). Moreover, c is also proven to be equal to t_{lag}, which is the lag time between the centroid of the rainfall excess hydrograph and the centroid of the direct runoff (Pedersen, Peters, and Helweg 1980) (Equation 2.5).

$$t_{lag} = c = t_Q - t_I \tag{2.5}$$

Where t_Q is the time between $t = 0$ and the centroid of the outflow and t_I is the time between $t = 0$ and the centroid of the inflow.

Note that the storage coefficient is often denoted as K in common theory. However, to differentiate this parameter with another k that shall be introduced in subsection 2.2.2, we will refer to it as c.

Figure 2.2 shows a plot as an example of a linear relationship between the storage and outflow of a system.

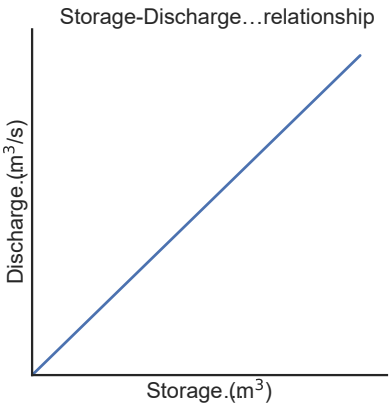

Figure 2.2: Example of a linear SQ relationship with $\alpha = 1$

Given that the slope of the plot is α, the value of c is then calculated as the inverse of α, meaning:

$$\alpha = \frac{1}{c} \tag{2.6}$$

Replacing Equation 2.4 into Equation 2.1, we have:

$$I(t) = c\frac{dQ(t)}{dt} + Q(t) \tag{2.7}$$

It can be seen that when the relationship between S and Q, i.e c, is known, then Equation 2.7 is solved and the outflow Q can be obtained. In theory, the derivation of c can be computed as t_{lag}, which means that it is possible to calculate c from the observed data of excess rainfall and the resulting direct runoff of a gauged catchment (Equation 2.5) (Pedersen, Peters, and Helweg 1980). The assumptions to derive c in this case are:

- The storms have to be isolated from each other time-wise
- The storms must be made of one well-defined peak only
- The storms are distributed uniformly all over the catchment

When the relationship between rainfall and runoff is linear, the value of c would be constant for a watershed regardless of rainfall. Nevertheless, the determination of c is not always straightforward, since it had been proven that for the same watershed, c changes as a function of rain characteristics (Sarma, Delleur, and Rao 1973).

2.2.2 Nash's Linear Reservoir Cascade

One of the most popular method of approximating discharge in applied hydrology is the Nash (1957)'s linear reservoir cascade approach. This method proposes a conceptual instantaneous unit hydrograph (IUH) containing two parameters, k

and n. The concept of unit hydrograph and IUH can be found in Appendix A. As described in section A.2, the effective rainfall for the IUH has to be instantaneous as well. Nash's conceptual model describes the effect of lamination of the rainfall by assuming that the process happening in a catchment is similar to the effect of routing the rain through a cascade of n identical linear sub-reservoirs, of which each possesses the same storage coefficient k. In this model, the instantaneous rainfall is placed in the sub-reservoir that is furthest from the outlet (Equation 2.9). After that, the flow is routed through all n sub-reservoirs, by equating the inflow of each sub-reservoir to the outflow of the previous one in series (Equation 2.10). A visualization of the concept is present in Figure 2.3. Follows the mathematical description of the model:

$$S_i(t) = k\, Q_i(t) \quad i = 1, 2, \ldots, n \tag{2.8}$$

$$\frac{dS_1(t)}{dt} = I(t) - \frac{S_1(t)}{k} \tag{2.9}$$

$$\frac{dS_i(t)}{dt} = \frac{S_{i-1}(t)}{k} - \frac{S_i(t)}{k} \quad i = 2, \ldots, n \tag{2.10}$$

where $S_i(t), Q_i(t)$ are the storage and outflow, respectively, of the i-th sub-reservoir at time t, $I(t)$ is the inflow to the sub-reservoir and $dS_i(t)/dt$ is the change of storage with respect to time for the i-th reservoir. Equation 2.8 specifies that each sub-reservoir is linear. Equations 2.9 and 2.10 describe the system of differential equations that can be used to determine the storage and outflow of the reservoir for every t, given initial conditions and inflow.

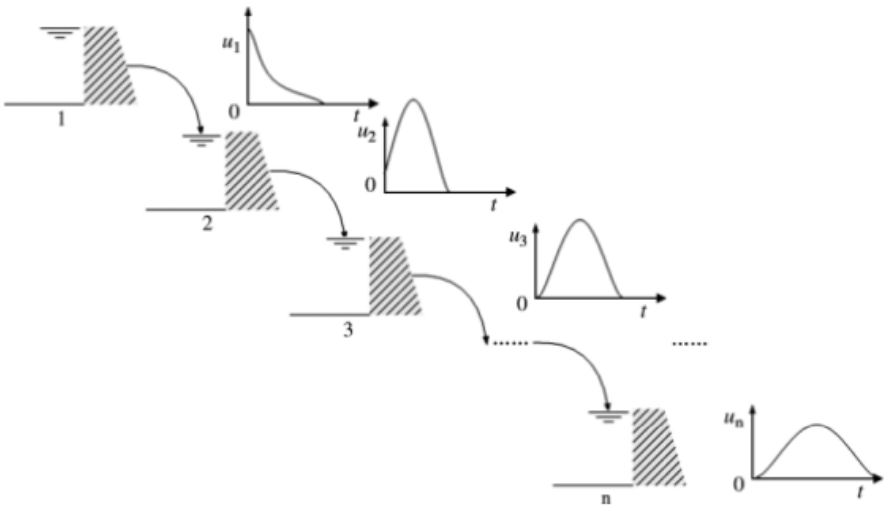

Figure 2.3: Concept of Nash's linear reservoir cascade (Li et al. 2008)

Given the description of the system, the total storage of the cascade reservoir at each time step is:

$$S(t) = \sum_{i=1}^{n} S_i(t) \tag{2.11}$$

where $S_i(t)$ is the storage of each linear sub-reservoir. Also, the discharge out of the system is the discharge of the last linear sub-reservoir in the chain and is calculated using the equation below:

$$Q(t) = \frac{S_n(t)}{k}, \tag{2.12}$$

where the linearity of the sub-reservoir and the fact that all sub-reservoirs have the same storage coefficient k are used.

2.2.2.1 Characteristics of Nash Model

Given an instantaneous inflow and $S_i(0) = 0$ for al sub-reservoirs as initial conditions, the system of differential equations previously defined can be solved, with the ordinates of the Nash IUH as follow:

$$u(t) = \frac{1}{k\,\Gamma(n)} \left(\frac{t}{k}\right)^{n-1} e^{-\frac{t}{k}} \tag{2.13}$$

where k and n are Nash's parameters, t is time and $\Gamma(n)$ is the gamma function that can be expressed as $\Gamma(n) = (n-1)!$. For the values of n that are different from an integer, $\Gamma(n)$ can be obtained by using the table of gamma function (Ven Chow 1988).

To calculate the time to the peak t_p as well as the peak discharge value Q_p, we can solve the equation $dQ/dt = 0$ and the results are:

$$t_p = (n-1)k \tag{2.14}$$

$$Q_p = \frac{1}{k\,\Gamma(n)}(n-1)^{n-1}e^{1-n} \tag{2.15}$$

Another parameter of the Nash IUH is the lag time t_{lag}, which is the time difference between the centroid of the inflow and discharge. This is the overall lag time distributing over all the reservoirs, which is different from the delay time k for each reservoir. (V. Singh 1992).

$$t_{lag} = kn \tag{2.16}$$

Figure 2.3 also shows that the higher number of linear reservoirs, n, there is, the flatter the peak and more delayed the discharge is from each reservoir. It means that increasing the number of n can cause a delay effect between the rainfall and the discharge. Additionally, with the same number of reservoirs n, if the delay constant k for each reservoir increases, the system's outflow also gets delayed. Therefore, it can be said that both parameters k and n are capable of delaying the system's outflow, as can be seen in Figure 2.4.

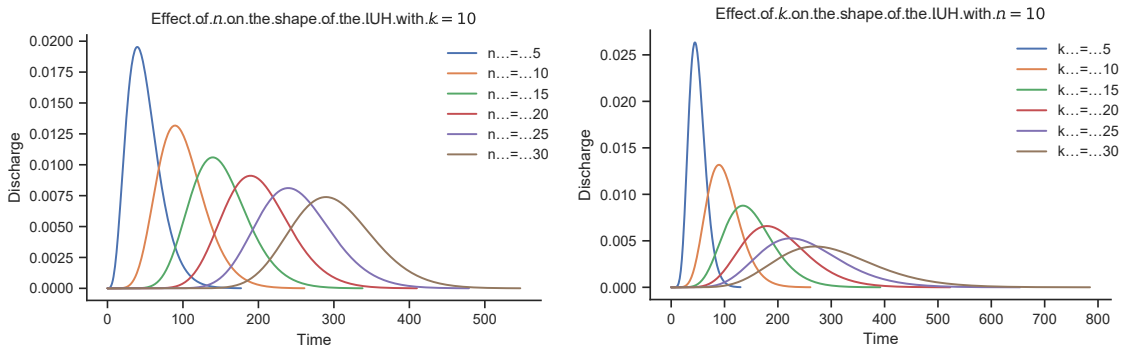

Figure 2.4: Effect of changing k (left) and n (right) on the Nash's IUH

Nevertheless, the effect of each parameter is more pronounced together when considering the case of different IUH with the same peak time t_p as in Figure 2.5. In this case, when n increases and, as a consequence, k decreases, the IUH tends to become narrower with higher peaks.

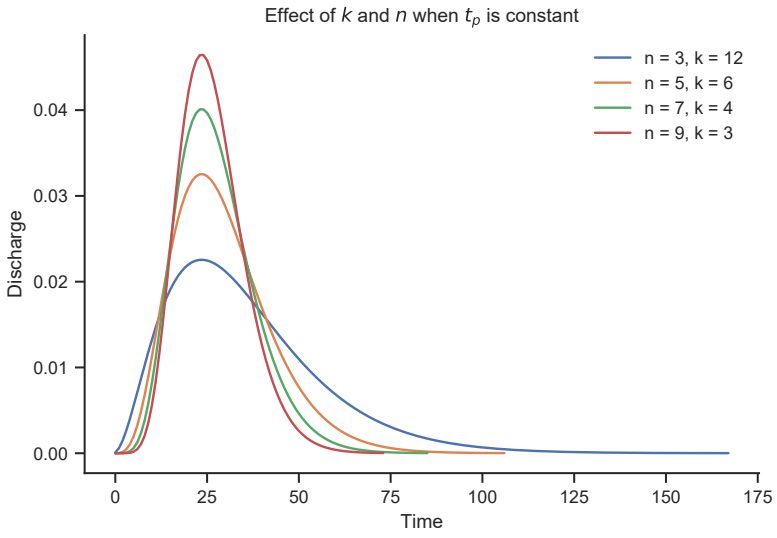

Figure 2.5: Effect of k and n on the shape and magnitude of IUH when peak time is constant

2.2.3 Other SQ Models

As mentioned above, the discharge in the system can be any function of storage (Equation 2.3), which means that there are other forms of SQ relationship other than linear. In this case, it is not possible to apply unit hydrograph theory.

Kimura (1961) introduced his SQ relationship as the following equation, which implements a loss mechanism:

$$S = cQ^p, \tag{2.17}$$

where c and p are determined by the characteristics of each catchment. When $p = 1$, Equation 2.17 becomes the equation for linear reservoir.

Moreover, for a long narrow reservoir, the backwater effect often alters the SQ relationship, causing a loop. This phenomenon is called "storage hysteresis effect", in which one discharge corresponds to two values of storage, as can be seen in Figure 2.6.

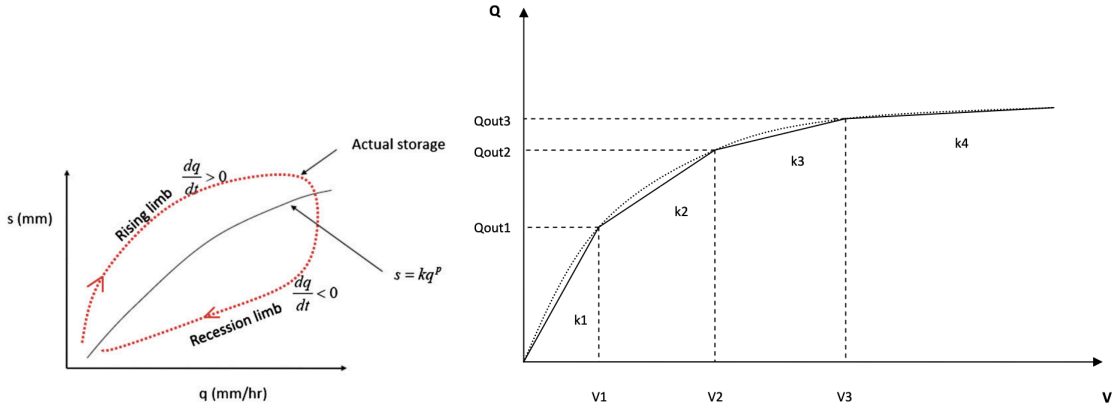

Figure 2.6: Illustration of loop SQ relationship (left) (Wu, Ho, and Yang 2011) and piecewise linear SQ relationship (right) (Kamradt 2008)

Equation 2.17 in this case cannot describe this effect. Therefore, it was proposed by Prasad (1967) the following equation:

$$S = c_1 Q^p + c_2 \frac{dQ}{dt}, \tag{2.18}$$

where c_1 and c_2 are dependent on the catchment's characteristic and the added term $c_2(dQ/dt)$ serves the purpose of differentiating between the rising and recession limbs. This equation was also further modified by Wu, Ho, and Yang (2011) to better describe this effect. While being accurate, Equation 2.18 is complicated to solve.

Transforming from the original Nash (1957) model, another approach was taken to substitute the n cascade of linear reservoir using one single reservoir with a nonlinear SQ function. The function can be calculated based on the pipe's filling de-

gree. Then, since the Equation 2.1 must be solved in discrete time, this non-linear SQ function can be described as a piecewise linear approximation (Figure 2.6). This approach also has the advantage of simplifying and decreasing the number of differential equations needed to be solved, while enabling also a better conceptualization of the backwater effect according to Vanrolleghem et al. (2009).

2.2.3.1 SQ Models for Other Structures

In general, this SQ relationship is very helpful in modeling all elements of the watershed or the drainage network, such as describing a transport stretch, or basin outflow, combined sewer overflows, surface spill, or weir discharge, etc. (Thrysøe, Arnbjerg-Nielsen, and Morten Borup 2019). Therefore, it is an attractive approach since various structures of the drainage network can be modeled using the same model engine.

Figure 2.7: SQ functions used to simulate discharge from different structures of the drainage network (adapted from Lei 2020)

Figure 2.7 shows that the SQ function can be used rather flexibly, with each form being able to describe many different types of outflow. This is only a simple type of piecewise linear SQ function and it can be any form and shape. For structures such as a transport stretch or a basin, the discharge often happens immediately whenever there is storage in the system. However, for CSO, surface spill, or weir discharge, there needs to be some storage in the network, i.e. the system needs to be filled, for the spilling to happen.

3 Data

This chapter presents the main data used to as inputs to construct the conceptual models as well as the procedures of creating the two virtual cities.

3.1 Rain Data used in MU+

For the simulation of the city in MU+, rain with a period of 3 months is applied starting from October 1^{st} 2009 at 00:10, ending on December 31^{st} 2009 at 23:50 (Figure 3.1). This rain is applied to simulate both the transport stretch (subsection 4.1.1) and the two virtual cities in MU+ (section 3.5).

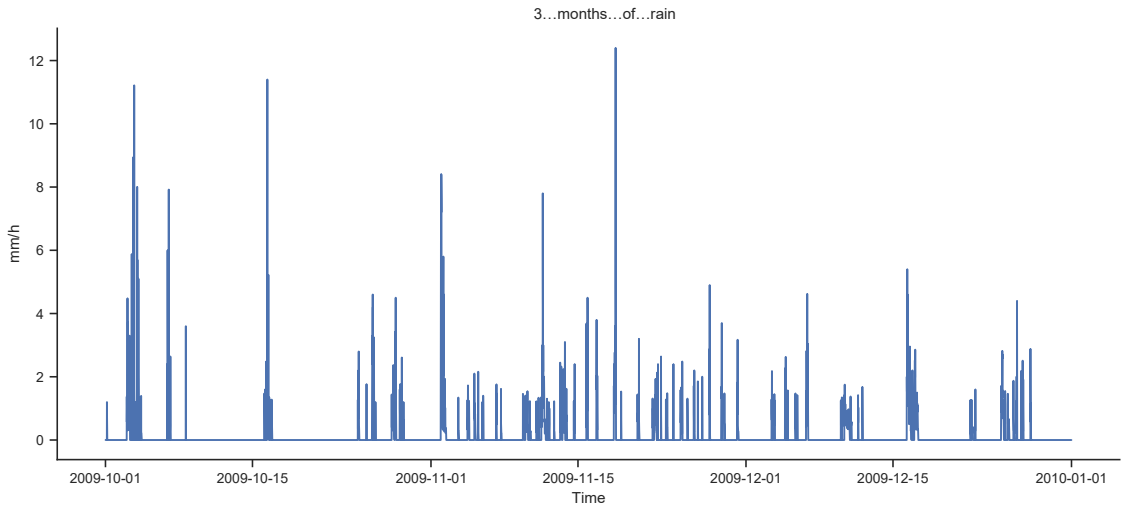

Figure 3.1: Rain time series 10-12/2009 used in the experiment

3.2 Dry Periods Exclusion

In the calculations for the transport stretch conceptual model, dry periods are excluded from the error computation to prevent skewing results. Dry periods in this study are defined as the time between rain events where no rain occurs for at least 10 hours. This inflow is obtained from MU+ when simulating a transport stretch with the rain in Figure 3.1, which is the inflow of all catchments into node G7, which can be seen more clearly in subsection 4.1.1. The dry periods that are excluded from calculations are presented in Figure 3.2. In this case, the conceptual model detects when there are dry periods in the inflow and doesn't calculate the error between MU+ and CM discharges at these times.

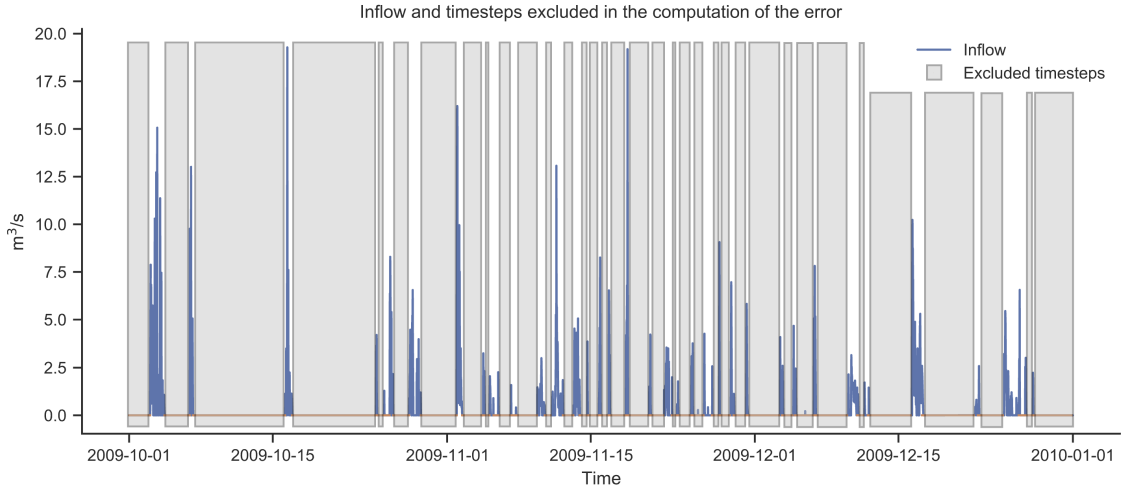

Figure 3.2: Inflow into G7 (blue) and time steps that are excluded from error calculation (grey)

3.3 Simulation Periods

Table 3.1 presents all the simulation periods and data used in this research. At different step of the modeling process, different simulation periods are considered due to technical limitations.

To find the storage constant c, the simulation of each storage and discharge pair is calculated using the inflow with excluded dry period similar to Figure 3.2. However, only 1000 minutes of inflow are computed instead of using 3 months of rain. The inflow used in this case can be seen in Figure 3.3.

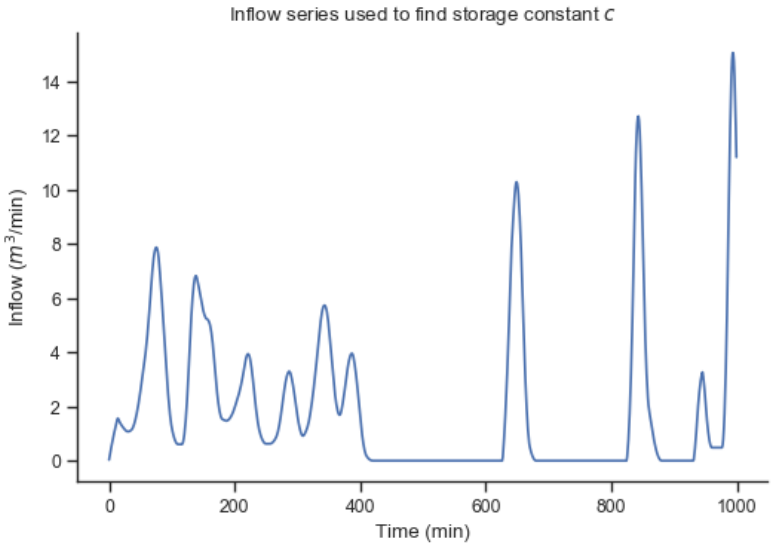

Figure 3.3: Inflow used to find storage coefficient c

Upon inputting the full inflow, i.e. like in Figure 3.2 but without dry period exclusion, errors arise from the differential equation system solver used in Python, because dry periods has too many $0s$ inflow values. This is the reason why the simulation of storage constant is done by excluding the dry periods. Nevertheless, only 1000 minutes are computed because the computational time is too expensive. We believe that 1000 minutes are enough to simulate the correct value of storage constant c and this value would not vary significantly regardless of which type of rain is being used as input. To test this, random inflow is used along with the excluded dry period inflow. Some examples of the results can be found in Appendix B.

Table 3.1: Summary of the types of data used in this research, how they are used and which sections in this study refer to the use of this data

Data	Uses	Sections
3 months rain $10-12/2009$	As input into MU+ model for both transport stretch and cities simulations to generate surface runoff (inflow)	3.5 4.1.1
Full inflow time series	As input to Nash model's transport stretch simulation to find k and n As input to optimize linear and piecewise linear SQ models	4.1.2.1 G.1 H
Inflow excluding dry periods	To calculate errors between discharges of MU+ and CM for transport stretch	4.1.2.1 4.1.2.3 G.1 H
1000 minutes of inflow excluding dry periods	As input to find storage constant c	4.1.2.2

3.4 Virtual Cities Configurations

Two city configurations were created, namely the Star and the Strip city, to explore the effect of different structures on the behavior of the urban drainage models. Both cities are virtually designed according to the outer contour shapes from Jia et al. (2019), where the Star city has the shape of a maple leaf while the Strip city that of a rectangle. The reason for using these shapes is that these are the two basic city shapes representing many urban areas. Having an artificial layout allows the exclusion of features that are only generated as a result of some specific locations' properties, hence making the models more representative and the response of different configurations more pronounced. Moreover, it enables better control of the system, where all parameters are designed completely from scratch without depending on a complicated real system where the lack of information might hinder a better understanding of the system's behavior. Besides, these basic cities can

make drainage systems more simplified, avoiding extra complexity as the underlying nature of this study is more of a proof of concept rather than an actual case study.

The drainage system is a combined system with both sewer and stormwater inflow designed using the Rational method mentioned in Appendix C. However, the sewer flow is only used for calculating pipe slopes and therefore disregarded later on as the project only focuses on stormwater management. The slopes of both cities are 0.01% with the direction from the top left corner to the bottom right as in Jia et al. (2019). The terrain is therefore nearly flat while this small slope still enables runoff to enter the drainage systems through manholes. A summary of the main parameters for designing drainage systems of the two cities is available in the following Table 3.2.

Table 3.2: Summary of main parameters for Star city and Strip city

	Star City	**Strip City**
Single house area (m^2)	360	
Large road width (m)	15	
Small road width (m)	7	
House runoff coefficient	0.4	
Road runoff coefficient	0.8	
Wastewater flow (l/p/s)	0.003	
Total area (ha)	24.7	21.3
Total population (people)	1032	960

3.4.1 Urban Drainage Systems Design

The designed cities are shown in Figure 3.4. When the catchment contains only 1 manhole, the catchment's runoff will flow in that manhole. On the other hand, when the catchment contains 2 manholes, the flow connections indicates which manhole receives the runoff.

Figure 3.4: Star City (up) and Strip City (down) design

Both cities have an outlet pipe that directs all the runoff out of the system. The details on parameters used to dimension manhole depths, slopes and pipe dimensions can be found in Appendix C. Overall, the designed system can withstand a storm of 2 years frequency without overflowing, while self-cleansing criteria are still maintained.

3.5 Mike Urban+ Cities Setup

After dimensioning the urban drainage network using the Rational Method, all information was imported to draw the cities in MU+. Both cities use Time-Area method to describe rainfall-runoff process and the simulation is run with a fixed 1 minute time step . The rain used in this experiment is also the same 3 months rain in Figure 3.1. After running the simulation for each city, the data of surface runoff for each catchment was extracted to be used as input for the conceptual model. The time series for outlet discharge of each city were also obtained as this is the point of comparison to evaluate how well the conceptual model performs.

4 Methods

In this chapter, the methodology used in this project is presented and explained. The objective of the study is to find the suitable SQ function to be used for the description of pipe flow and how to divide compartments to best describe the conceptual drainage network for different city configurations. To achieve this aim, two main conceptual models are built to simulate discharge from the transport stretch and from virtual cities. The main processes can be seen in Figure 4.1.

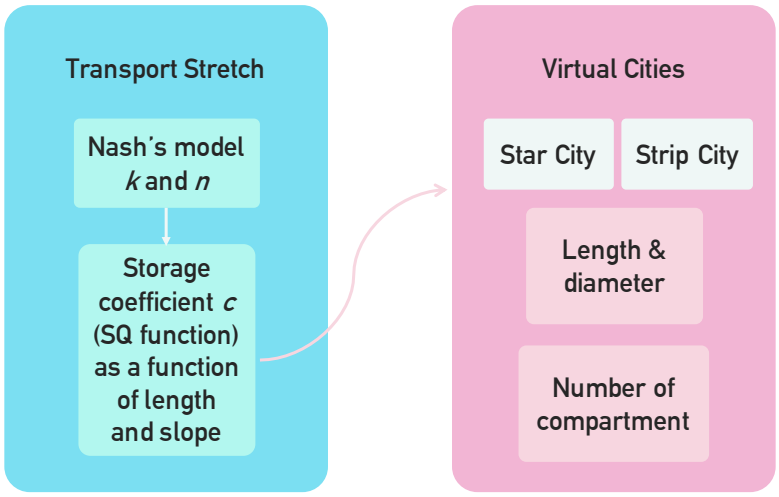

Figure 4.1: Flow chart of the main process

First, the purpose of the transport stretch model is to find a linear SQ function that can describe the discharge of a single pipe having certain salient physical characteristics (length and slope). Then, the resulting SQ relationships are used to simulate the two virtual cities. Here, decisions are made on how to choose the correct length and diameter that will be used to find the SQ function. The cities are split into a number of compartments in different configurations. Finally, the performance and behavior of each compartmentalization are evaluated.

4.1 Transport Stretch Conceptualization

The aim of this section is to find a linear SQ function that best approximates the behavior of a transport stretch with a given length and slope. The steps are as follow:

- 4.1.1 Simulate transport stretch with different length and slope combinations in MU+

- 4.1.2.1 Build CM to find best k and n for Nash model, by computing the root mean square error (RMSE) between the conceptual model's and the MU+'s discharges
- 4.1.2.2 Simulate storage-discharge values to find linear SQ relationship (storage constant c)
- 4.1.2.3 Apply the SQ function to simulate outflow from a transport stretch and compare outflows between CM and MU+

It is also possible to obtain the value of the storage constant c through direct optimization of the SQ slope without having to simulate through the Nash model. Nevertheless, the Nash model is more straightforward to be implemented as the first step to this study, in terms of coding for an inexperienced Python user, compared to optimizing SQ slope, which requires a deeper understanding of the code used in the *surrogate model engine for water networks by DTU* (M. Borup 2018).

4.1.1 Mike Urban+ Model for Transport Stretch

Firstly, a simple model in MU+ is set up using the collection system model with rainfall-runoff module, with Time-Area method applied. The MU+ model has only one pipe representing a transport stretch starting from a node called G7 and ending at the outlet. The entire city has an area of 24.7 ha with an average imperviousness of around 48% depending on the catchment (see subsection C.4.1). The layout of this experiment from MU+ is present in Figure 4.2.

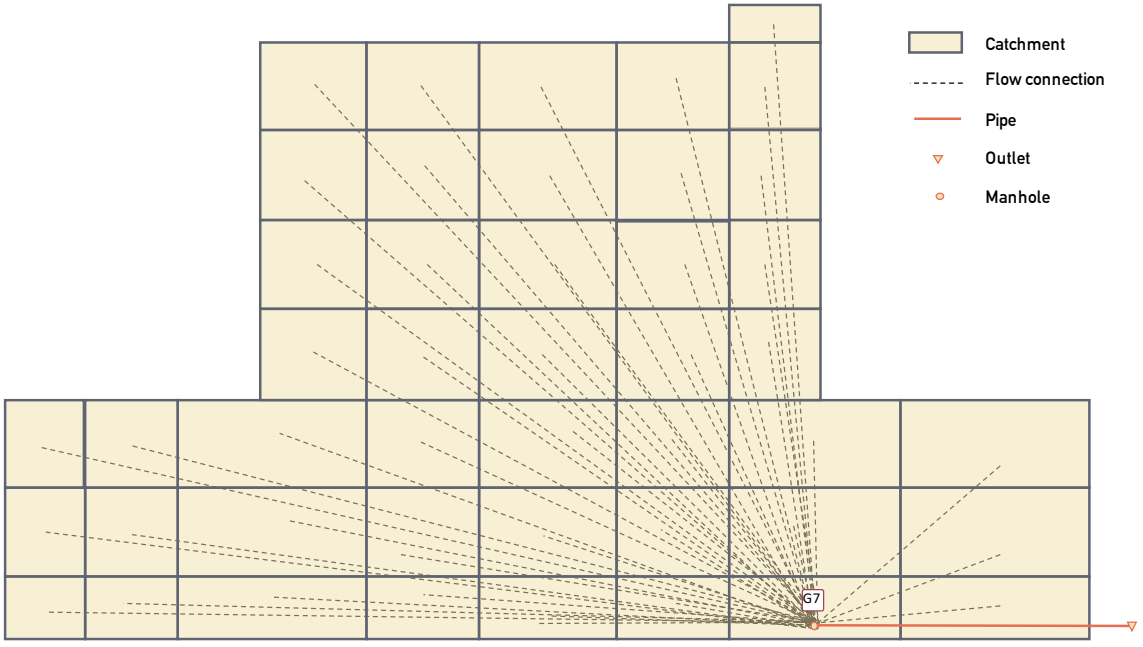

Figure 4.2: Transport Stretch Layout from MU+

The 3-month long rain from 10-12/2009 in Figure 3.1 is used and distributed all over the city. All runoff of the catchment is drained into node G7 and flows out of

the system at the outlet, meaning that node G7 is the single point of entrance for the transport stretch.

The length and slope of the pipe are then modified in MU+ to simulate the discharge. 110 simulations were done in which each length and slope were modeled. The values for length range from 50m to 500m with a step increase of 50m (10 lengths), while that for slope are between 1‰and 50‰with a step of 5‰(1‰; 5‰; 10‰... 50‰) (11 slopes). The diameter of the pipe is 2m to ensure that node G7 does not overflow and all the runoff upstream can enter the system and corresponding discharge at the end of the pipe is generated. Plus, with a large diameter, it ensures that water gets transported to the outlet without having any effect on the hydrograph. Additionally, the objective of this transport stretch conceptualization is to finally obtain the SQ function, meaning to model the transport delay with each length and slope of the pipe. This model does not aim at simulating the transport capacity of the pipe. Moreover, the diameter only determines the maximum flow, i.e. the cutoff point of the SQ function, which will be derived afterward in subsubsection 4.2.1.1.

The MU+ model is run with a fixed time step of 1 minute. For each run, the inflow time series into G7 and the discharge at the end of the pipe are recorded and saved to be used as input for the conceptual model. The point where the discharge time series is extracted is the last arrow before the outlet node, to ensure that regardless of the pipe length, the discharge is always recorded at the point nearest to the end of the pipe, where the delay effect is most highlighted. The reason why the discharge cannot be obtained directly at the outlet is due to MU+'s computational scheme, which alternates between the water level and water discharge calculations along the pipe. At the point of the outlet, only water level is reported, not the discharge.

4.1.2 Conceptual Models for Transport Stretch

This section describes all the conceptual models used to derive the storage constant c for the transport stretch based on each length and slope combination and simulate outflow based on c.

4.1.2.1 Nash's Linear Reservoir Cascade Parameters

This section presents the derivation of Nash's linear reservoir parameters k and n from the results of discharge obtained from MU+. The aim of this is to find the values of k and n for each pipe length and slope so that they can be applied afterward to simulate the storage and discharge and based on that obtain the storage constant c. This is done by testing different combinations of k and n for each set of pipe length and slope and selecting the best combination that has the least error compared to MU+. This conceptual model may also be referred to as conceptual Nash model later on.

The inflow time series from MU+ (inflow to node G7) is used as input to the con-

ceptual model and the MU+ discharge is later used to find the best combination of k and n from Nash's linear reservoir model, with the assumption that a transport stretch now acts as a reservoir. The inflow to G7 in this case is the full time series, which is the same as in Figure 3.2 but without any dry period exclusion. Missing values in the time series are filled by linear interpolation to make sure that the two time series are sampled at the same frequency.

A list of values of k and n are then defined to be tested, with n ranges from 1 to 300 with a step of 1 and k ranges from 10^{-2} to 10 logarithmically with a step of $10^{0.0005}$. The range of k is examined on a logarithmic scale because, after some initial runs of the model, it was noticed that k is often selected to be quite small. Hence, a logarithmic scale is more suitable for exploring such small values. Each combination of k and n is tested, which generates different instantaneous unit hydrographs that are then convolved with the inflow to simulate the conceptual model's discharge. This discharge is computed by using the Nash model applied to the discrete case as described in the section A.2.

The root mean square error (RMSE) is computed between the discharge from MU+ and the conceptual model's discharge to evaluate the combinations of k and n that results in the outflow of CM that is closest to MU+ model.

$$\text{RMSE}(D_{CM}, D_{MU}) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(D_{CM}(t) - D_{MU}(t)\right)^2}, \tag{4.1}$$

where D_{CM} is the discharge of CM, D_{MU} is the MU+ discharge and T is the number of time steps.

In this case, RMSE is only calculated between discharges of MU+ and CM when there is rain, i.e. with the exclusion of dry periods in the inflow, as in Figure 3.2. The aim of this is to avoid measuring the error for those time steps when there is no rain event, which might skew the results towards selecting the values of k and n that generates the instantaneous unit hydrograph to model the unwanted dry period. In other words, the main aim of the CM is to simulate correctly the peaks of rain. Therefore, by excluding long dry periods, the model will focus more on generating correctly the rain events. This results in a slightly modified version of the formula previously shown that finds the RMSE. For each iteration of k and n pair, the RMSE is stored and the combination with the smallest error is chosen.

The model is implemented in Python and the code for the transport stretch Nash model optimization is available in section I.2. The initial version of the script was provided by Löwe (2020a).

4.1.2.2 Storage Coefficient

When k and n pair for each pipe length and slope are found, it is possible to solve the system of differential equations from the Nash model to derive the values of

storage and discharge at each time point and then obtain the storage constant c from the slope fitted.

The process of finding storage coefficient c involves solving the system of differential equations that are used to define the Nash model using `odeint` from the `sklearn` package in Python script (section I.3) based on the script given by Löwe (2020b). This code script can generate the set of differential equations automatically based on the found k and n, and subsequently simulates the Nash cascade of reservoirs in a way that the relationship between storage in and discharge out of the cascade can be derived.

For each pair of length and slope, the best parameters found for n and k (obtained as the results of subsubsection 4.1.2.1) are used to define the number and the storage constant of the linear sub-reservoirs respectively. The storage of all sub-reservoirs is assumed to be initially empty. The inflow used is the one recorded in the experiment run with length 50m and slope $1‰$, as the inflow is the same for all pairs and the only differences present are due to numerical approximation. In this case, only 1000 time steps of inflow are used as input instead of the whole rain series as can be seen in Figure 3.3.

Given the properties described in subsection 2.2.2, a plot of the relationship between storage and outflow can be drawn using the inflow from Figure 3.3 and system of differential equations (Equation 2.9, Equation 2.10). Then, a linear fit with no intercept is used to find the slope $1/c$ of the relationship between Q and S. Using the slope, the storage constant c of the linear reservoir is obtained by taking its inverse using Equation 2.6.

4.1.2.3 Discharge Simulation Based on SQ Function

When the values of storage constant c are known, the discharge of the conceptual model from the linear SQ model is simulated for validation against the MU+ discharge. This section describes the steps taken in computing the discharge from a linear transport stretch with storage constant c.

The model for the transport stretch conceptualization uses the *surrogate model engine for water networks by DTU* (M. Borup 2018) to define a simple network which constitutes of a single connection to the outlet. This connection will then behave according to the linear SQ function described by the storage constant c, i.e. having slope $1/c$ (as a result of subsubsection 4.1.2.2). The maximum flow q_{max} that the SQ function can reach is set at 200 m^3/s, an extremely high value describe a linear SQ function without having any flow limitation. The reason is that the MU+ model also simulates flow in a pipe with 2m of diameter, in order to not affect the outflow hydrograph. Therefore, when running the conceptual model using SQ function, the maximum flow is set high to impose no effect on the discharge achieved.

The inflow of the model will then be the same as the one for the Nash model. The discharge from the pipe is measured at the outlet and the simulation performance

is evaluated using the same technique used in the Nash model, which is the RMSE (Equation 4.1) calculated neglecting the errors computed during dry periods as in Figure 3.2, to provide a fair comparison.

The discharge from the SQ conceptual model is run using Python, whose codes are available in section I.4 and section I.5, applied as a linear SQ function.

4.2 Virtual Cities Simulations

When the SQ functions of the transport stretch of different lengths and slopes are computed, it is possible to translate this into simulating the cities' discharge. The city is lumped into compartments acting like a transport stretch, whose discharge is represented by an SQ function. In order to correctly choose the SQ function to describe a drainage network, a number of decisions need to be made:

- How should the length of transport stretch be chosen in the compartments to describe the delay effect accurately?
- How to choose the correct diameter to capture the flow limit?
- How should extrapolation be done to find the SQ functions when the length exceeds the test range of 500m in subsection 4.1.1?
- How big should a compartment be?

To answer these questions, a number of scenarios are created where the conceptual model is parameterized in different ways. The description of these scenarios will be presented in the following sections.

4.2.1 Conceptual Cities Setup

The conceptual model for the cities is implemented using the *surrogate model engine for water networks by DTU* (M. Borup 2018). A modular approach has been taken. Specifically, the cities are described by means of a network structure, each pipe connects one manhole to another with a direct link. This allows a straightforward definition of a compartment as a subnetwork, i.e. the subset of manholes and the pipes that connect them to each other.

Each manhole collects runoff from a catchment, as described in Figure 3.4. When a compartment is defined, its inflow will be given by the sum of the runoffs from all the catchments that are linked to manholes present in the compartment. The runoff of each catchment is obtained from the simulation of MU+. It is possible for the conceptual model to compute the runoff to each catchment without having to use the results from MU+. However, the aim of this study is mainly to simulate discharge correctly. Therefore, using the same inflow will eliminate a potential source of deviation between the MU+ and CM, so that the effect of compartmentalization is the most highlighted.

Then, each compartment is modeled as a transport stretch, according to its physical characteristics, meaning that the network of pipe inside the compartment is now lumped into 1 transport stretch that can be described using the SQ relationship derived from subsubsection 4.1.2.2.

The code written for creating the conceptual model in Python (section I.4 and section I.6) of the cities allows for fast prototyping of the compartments. Once the city structure is defined and the physical characteristics of the pipes are provided, each compartment can be quickly described as a list of manholes and the code will autonomously select the inflows and compute the SQ functions describing each compartment.

4.2.1.1 Methods of Choosing Length and Diameter for Virtual Cities

The parameters to be decided are the length and diameter of each compartment since the parameter length decides the discharge hydrograph and the diameter affects directly the maximum flow allowed in each compartment. Here, several choices have been made.

Weighted average of the lengths. The length is given by the average of the lengths of each pipe in the compartment weighted by the area of the catchment whose runoff flows into the pipe.

Sum of longest transport stretch. The length is given by the total length of the longest continuous transport stretch in the compartment.

Weighted average diameter. The diameter is computed by taking the weighted average based on the catchment area, as it has been done for the length.

Maximum diameter. The diameter is computed by taking the maximum amongst the diameters of each pipe in the compartment.

4 combinations of length and diameter are tested to understand which set of parameters works best at describing the cities' discharge. Then, for ease of analysis in this thesis, only the best combination is used for further analysis of different levels of compartmentalization.

The slope of the compartment will be **weighted average slope** based on the area of the catchment.

The **maximum flow** of the catchment is computed by using the Manning equation and using the parameters for the pipe just found:

$$q_{max}(N) = \frac{1}{m_c}\pi \left(\frac{D}{2}\right)^2 \left(\frac{D}{2}\right)^{\frac{2}{3}} \sqrt{S} \tag{4.2}$$

where N is the compartment, m_c is the Manning coefficient of the pipe, which is chosen to be 0.011 similar to that in MU+, $\pi \left(\frac{D}{2}\right)^2$ is the area of the pipe, S is the slope (dimensionless). q_{max} is the upper limit of discharge in a pipe to limit the

flow when using the SQ function. The relationship between storage and discharge when q_{max} is applied is described in Figure 4.3.

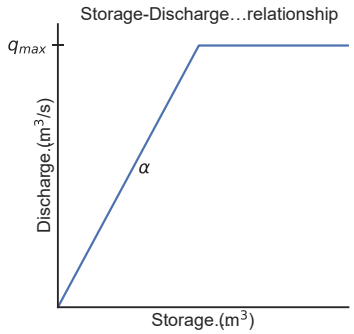

Figure 4.3: Plot of SQ function with q_{max}

4.2.1.2 Model to Find the Slope of the SQ Function

Once the parameters describing the compartment are found, the correct slope of the SQ function needs to be identified. The slope of the SQ function $(1/c)$ is computed only for pipes having the slopes and lengths tested. This is not an issue when the values of length and slope are within the ranges tested of 50m to 500m for the lengths and $1‰$ to $50‰$ for the slopes. However, when the length and the slope of a transport stretch exceed these limits, extrapolation is required to compute the SQ function. The reason behind having an extrapolation model is that when optimizing the transport stretch, it was not known that the lengths used to describe a city compartment would exceed 500m. At the same time, the time constraint did not allow redoing the optimization process for the transport stretch with longer lengths.

The computation is obtained based on the results of the storage constant c in the tested range. From there, the extrapolation model is fitted using least squares regression to produce an equation that can best predict the storage constant c. The resulting extrapolation model can be seen in subsubsection 5.1.2.1.

When the values are within the range of the already simulated length and slope, the value for the slope of the SQ relationship is found by locating the closest pair of length and slope to the ones requested and retrieving the value associated with it, which is the result of subsubsection 4.1.2.2.

4.2.1.3 Conceptual Model Evaluation

To evaluate the quality of virtual cities' conceptual models, it is necessary to use a combination of different performance measures. For evaluation of the general model performance, Nash-Sutcliffe efficiency (NSE) is used. To assess the peak, three measures are taken into account: PEP is the percentage error in peaks, $PDIFF$ is the difference in magnitude of the peaks and $PTDIFF$ is the difference in the time that the peaks occur between MU+ and CM. The evaluation is

done for each rain event during the 3 month long rain. The events are identified by analyzing the discharge of the MU+ model and selecting those periods where continuous evident discharge occurs. To define a significant rain event, minimum duration and volume discharged are used as criteria. A rain event is arbitrarily characterized by discharging at least 100 m^3 of outflow in total while lasting for a minimum of 20 minutes. Once the events are identified in the MU+ discharge, outliers caused by mismatches in peaks with the CM discharge are removed to provide a correct evaluation of the performance of the different models. The same events are used across scenarios in a given city, to accommodate different discharge patterns between cities.

$$NSE = 1 - \frac{\sum_t \left(q_{CM}(t) - q_{MU+}(t)\right)^2}{\sum_t \left(q_{MU+}(t) - \overline{q}_{MU+}\right)^2} \tag{4.3}$$

$$PEP = 100 \frac{\max_t(q_{i,MU+}(t)) - \max_t(q_{i,CM}(t))}{\max_t(q_{i,MU+}(t))} \tag{4.4}$$

$$PDIFF = \max_t(q_{i,MU+}(t)) - \max_t(q_{i,CM}(t)) \tag{4.5}$$

$$PTDIFF = t_{peak,i,MU+} - t_{peak,i,CM} \tag{4.6}$$

where $q_{MU+}(t)$ is the discharge from the MU+ model, the $q_{CM}(t)$ is the discharge from the conceptual model, $q_{i,MU+}(t)$ and $q_{i,CM}(t)$ are the discharges from the respective models for the i-th rain event while $t_{peak,i,MU+}$ and $t_{peak,i,CM}$ are the time to the peak for the i-th rain event. The average discharge of the MU+ model used in the NSE is given by $\overline{q}_{MU+}$.

Ideally, the NSE value should be close to 1, with $NSE = 1$ indicates a perfect fit. A good model should have NSE of at least 0.8. On the other hand, all peak performance measures should be as close to 0 as possible.

4.2.2 Virtual Cities Compartmentalization

When the parameters to describe the compartments are obtained, it is possible to apply them to examine the different ways of compartmentalization of the cities. This section describes different levels of compartmentalizing the two virtual cities. Each city has 4 levels, starting with coarse to medium, fine, and full network. The coarse level lumps the entire city into 1 compartment, hence will be described using only 1 SQ function. The full network level indicates that every single pipe in the network is computed, meaning that there is no lumping and this is the most detailed level. Table 4.1 shows the spatial extent of each level of compartmentalization for the two virtual cities.

Table 4.1: Area ranges of different compartmentalization levels

Compartment level	Area (ha)
Coarse	> 15
Medium	$7 - 15$
Fine	< 7

The full parameters of each compartment in each scenario of each city can be found in Appendix D. The medium and fine levels will be illustrated in the following sections.

4.2.2.1 Star City

The medium level of compartmentalization for Star city can be seen in Figure 4.4. The arrows in these figures describing the level of compartmentalization indicate the flow of each compartment. The compartments are colored based on which catchments are included in them. In this case, there are mainly 3 compartments and a transport stretch from G6 to G9. This is called a transport stretch to highlight its function to create more delay before compartment 1 and 2 flows to the outlet.

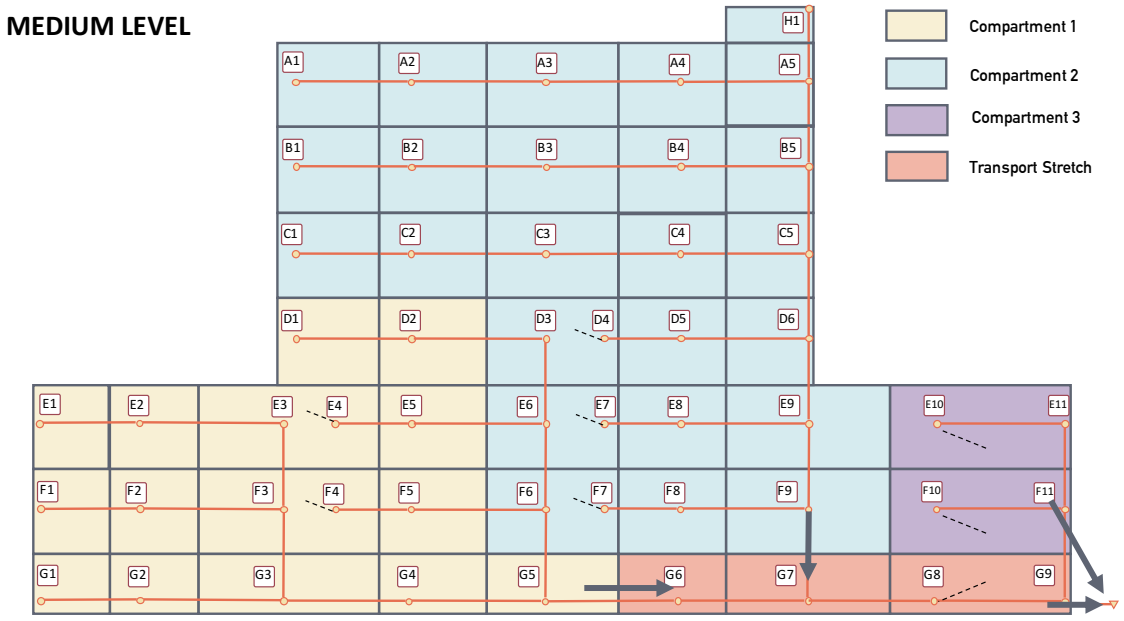

Figure 4.4: Medium level of compartmentalization for Star city

The fine level divides the Star city into 7 compartments in total as shown in Figure 4.5.

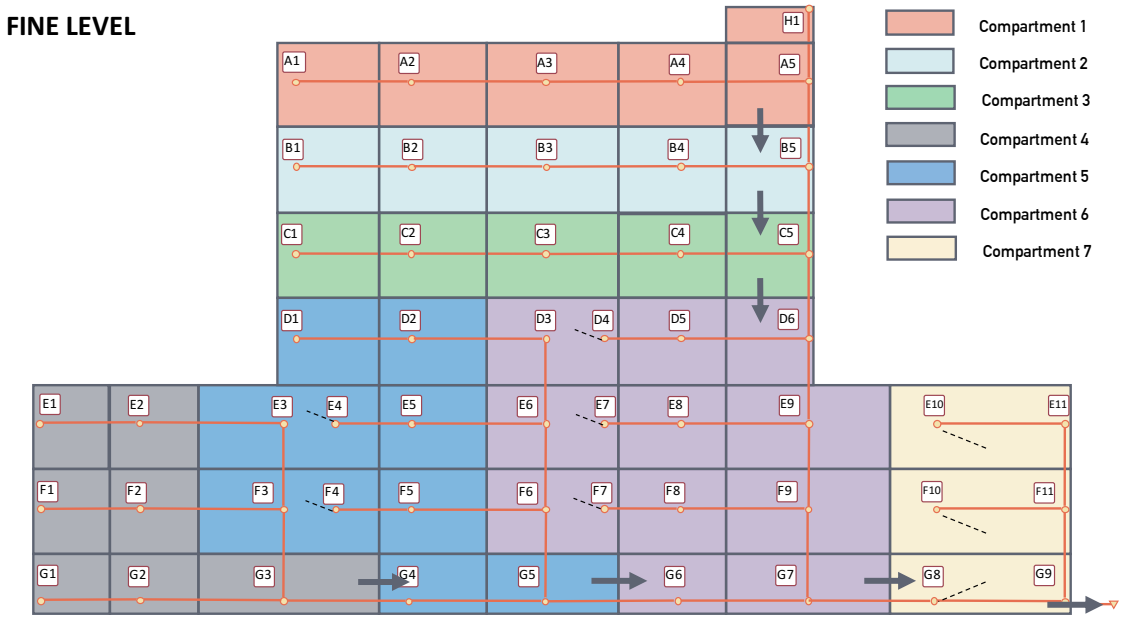

Figure 4.5: Fine level of compartmentalization for Star city

4.2.2.2 Strip City

For the Strip city, the medium level divides the city into 2 compartments (Figure 4.6) while the fine level splits the city into 6 compartments as in Figure 4.7.

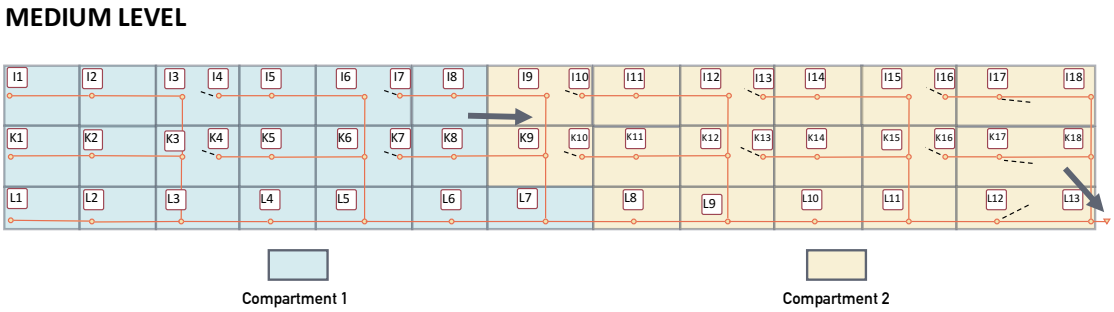

Figure 4.6: Medium level of compartmentalization for Strip city

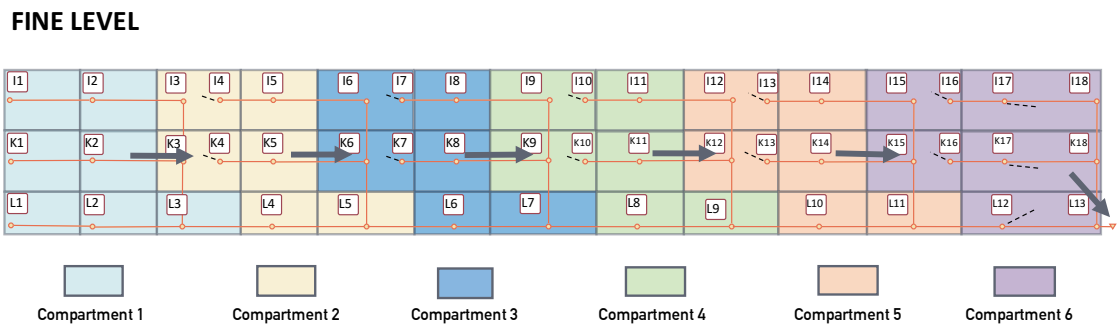

Figure 4.7: Fine level of compartmentalization for Strip city

5 Results

In this section, the results of the conceptual models are presented. It consists of two main results corresponding to two main CMs:

- Transport stretch conceptualization
- City compartmentalization

5.1 Transport Stretch Conceptualization

This section describes and analyzes the results achieved from the process of conceptualizing a transport stretch based on different combinations of length and slope. It includes these following contents:

- Linear reservoir cascade parameters (k and n values)
- The derived storage constant c
- Discharge of the transport stretch using storage constant c (SQ relationship)

5.1.1 Linear Reservoir Cascade Parameters

5.1.1.1 Values of k and n

The values of k and n from each combination of length and slope that best represents a single transport stretch (smallest RMSE compared to MU+ outflow) can be seen from Figure 5.1. The values of the delay constant k for Nash 1957's model here is in the resolution of minute, not second, since the time step tested in the conceptual model is in minute.

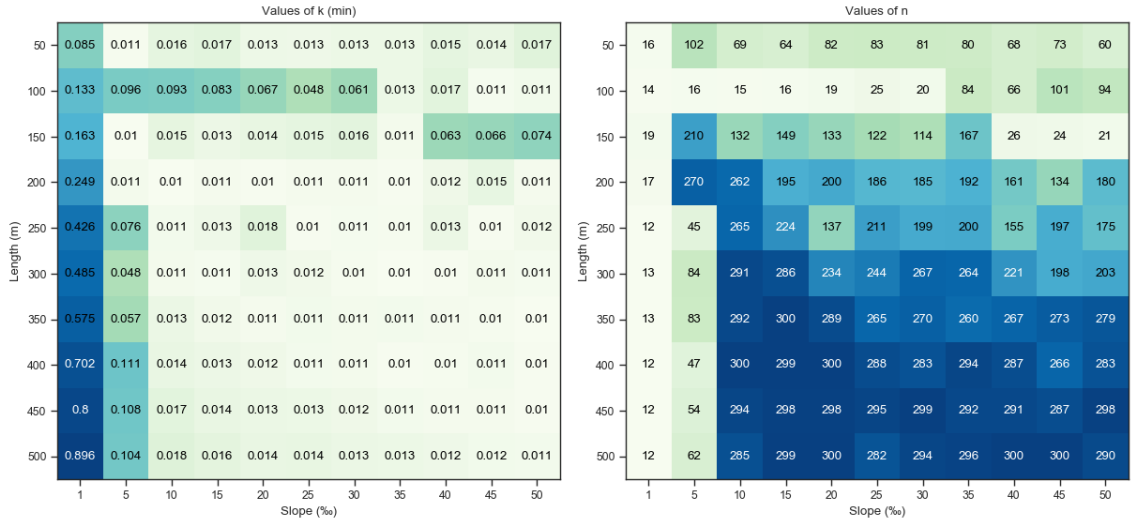

Figure 5.1: Values of k (minute) and n for each slope and length

From Figure 5.1, it is not possible to see a trend in either the values of k or n alone. However, it can be observed that these two parameters correspond to each other, meaning that wherever there is a high value of k, there is a correspondingly low one for n.

The biggest k is only 0.896 minutes while the highest n reaches 300, which is the highest value tested. This means that the model reaches the biggest n in the test range. The values of k and n are expected to follow a clearer trend, determined by the geometry of the pipe. With short and steep pipes the delay will be small, caused by the time that it takes for the water to flow through the pipe (which is a function of both length and slope). The shape of the discharge peak is mostly determined by the acceleration the water experiences inside the pipe, with bigger acceleration corresponding to narrower peaks, a phenomenon happening in pipes with high slopes and short lengths. In other words, the pipes that are longer and flatter would have more delay and the discharge peaks would be shallower. This results in our assumption that there should be a consistently increasing value of n and decreasing k with increasing pipe length and slope to simulate the sharper peaks. Overall, the model does choose high n and small k values for these pipes, but chosen values also do not follow any apparent trend.

Nevertheless, as mentioned before, even with the values of k and n seeming random, they correspond to each other, suggesting that these two parameters cannot be considered separately but must be taken into account together. When considering a constant time to the peak t_p, with bigger n and smaller k, the resulting IUH will have the same delay but increasingly sharp peak. It means that the model purposefully chooses these k and n values in order to describe closely the sharp and short-duration peaks of discharge happening mostly in pipes of bigger slopes.

5.1.1.2 Nash Model's Lag Time and RMSE

Figure 5.2 shows the duration in minute of the time it takes from center of mass of the instantaneous rainfall to the center of mass of the IUH, t_{lag}. This value is computed from Equation 2.16. Here, it is clear that there is an obvious trend when taking into account both k and n as mentioned before. The higher value of t_{lag} indicates the more time the discharge is delayed compared to the inflow. This trend is reflected accurately in our case, where t_{lag} increases with longer pipe stretch and lower slope, illustrating clearly that it always takes longer for water to travel through the pipe when the pipe is long and flat. Once again, this suggests that the parameters k and n must be evaluated simultaneously to understand how the model works. Figure 5.2 also suggests that the delay in discharge is influenced more significantly by the pipe length and less by the slope, which is also similar to our assumption.

The RMSE found between the simulated discharge of the Nash model and the MU+ model also shows a trend (Figure 5.2). Since the RMSE values are relatively small, the presented values in Figure 5.2 are multiplied by 10^3 for easier interpretation.

Figure 5.2: Lag time (min) of the instantaneous unit hydrograph (left) and root mean square errors (m^3/s) multiplied by 10^3 of different lengths and slopes for the Nash model (right)

It is evident that the Nash model has more difficulty simulating longer and flatter pipes. The reason for this bigger error can be seen in Figure 5.3.

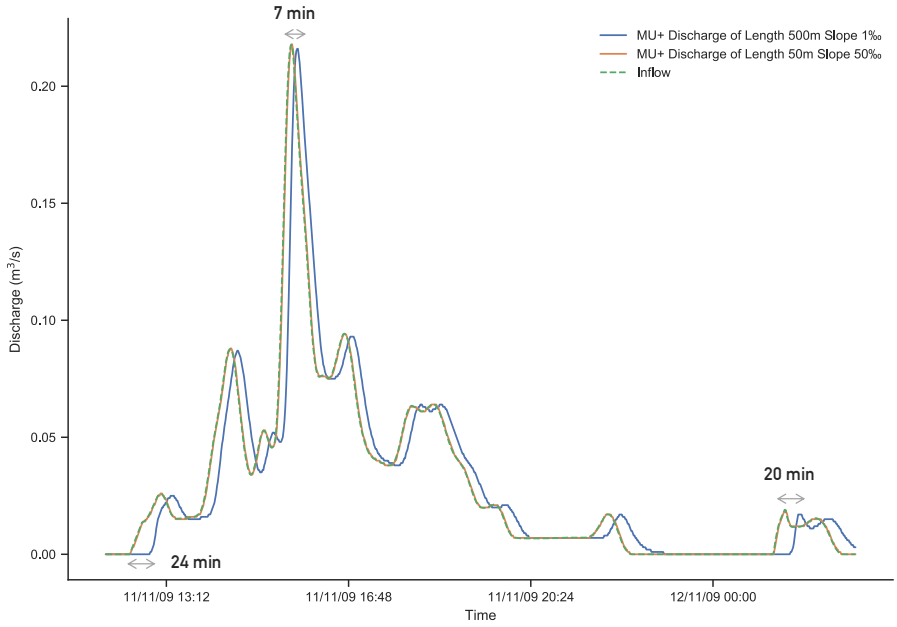

Figure 5.3: MU+ inflow and discharge of transport stretch length 500m, slope 1‰ and length 50m, slope 50‰

Figure 5.3 shows the discharge simulated from MU+, which is considered as the "true discharge" to calibrate the conceptual model against. Upon observing how the discharge is delayed compared to the inflow, it is clear that for the shorter and

steeper pipe (length 50m slope 50‰) has a discharge that has nearly and consistently no delay or difference compared to the inflow. This results in the fact that the Nash model can much more easily find a correct set of k and n to match with the MU+ discharge, hence the RMSE for this case is the lowest. On the other hand, the MU+ discharge for the long and flat pipe evidently has some delay compared to the inflow. However, closer inspection of the delay for this case indicates that the lag time is not constant. For example, the sharp and high peak has a much less delay between the discharge and the inflow (7 minutes) compared to the flatter peaks (24 and 20 minutes). It means that the MU+ discharge does not consistently delay the discharge by a certain amount of time compared to the inflow, hence indicating a non-linear effect in the "true discharge" itself for long pipes. Therefore, the conceptual model cannot replicate this non-linear delay accurately, resulting in the higher RMSE.

5.1.1.3 Discharge Comparison Between Nash Model and MU+

This section presents the results of using the chosen k and n combination from the conceptual Nash model on simulating the discharge of the transport stretch in comparison to the MU+ discharge. Figure 5.4 indicates the comparison between conceptual Nash model and MU+ outflows for the different transport stretches.

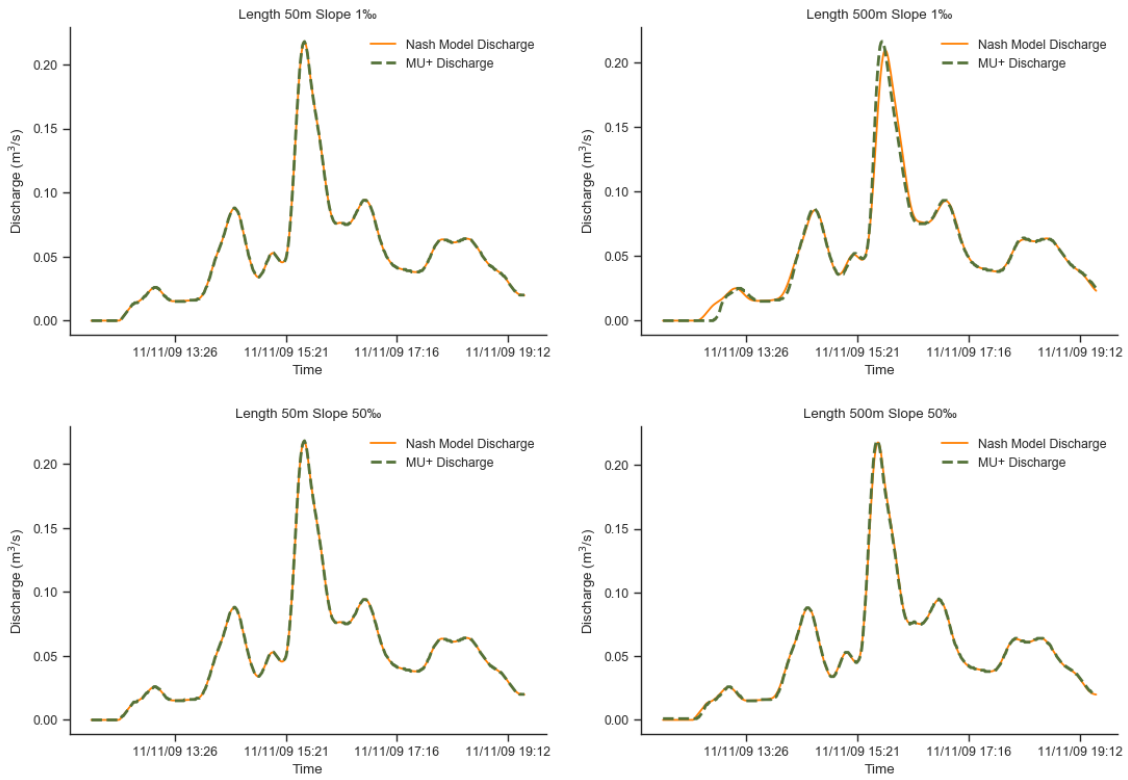

Figure 5.4: MU+ and discharge comparison between Nash model and MU+

It can be seen that for most pipe lengths and slopes, there is nearly no difference between the Nash model and the MU+ one, except for the long and flat 500m pipe

with 1‰ slope. The Nash model's discharge in this case did not match exactly both the delay time and the magnitude for the discharge at the peaks. For the high and sharp peak, this discharge arrives slower and lower compared to the MU+ outflow, while the opposite happened for the flat and smaller peaks. This issue was already addressed in subsubsection 5.1.1.2, resulting in a high value for RMSE. For shorter pipes or steeper pipes, the Nash model works relatively well at simulating discharge closely to the results from MU+, corresponding to the lower values of RMSE.

5.1.2 Storage Constant c

The storage and discharge of each pipe length and slope combination are simulated based on the system of differential equations as described in subsubsection 4.1.2.2. Figure 5.5 presents the SQ relationships of 4 slope and length combinations. The SQ plots for all 110 combinations can be found in the Figure E.1.

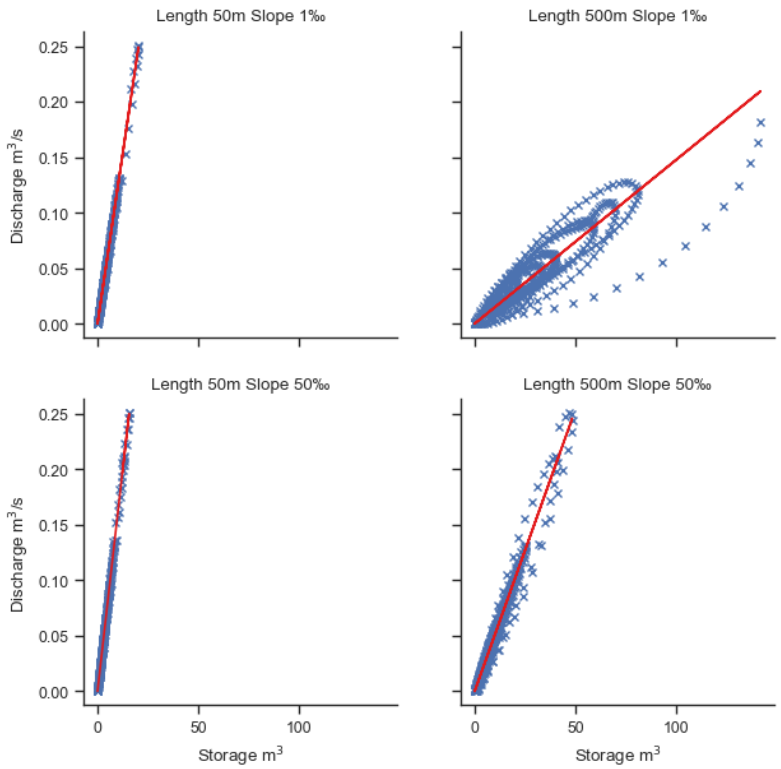

Figure 5.5: Storage and discharge simulations based on values of k and n with linear SQ slope fit (in red)

Figure 5.5 demonstrates also a clear trend for the relationships of storage and discharge based on different lengths and slopes. With shorter transport stretches, the relationship is mostly linear, where the outflow increases with increasing storage. Nevertheless, when the pipe gets longer, i.e at 500m, the SQ relationship is not a straight line anymore but resembles a loop, similar to the hysteresis described in Figure 2.6. The lower values of discharge happens at the beginning of the rain

while the higher ones are created at the end of the rain event. First, there is an increase in storage in the beginning when water starts flowing in the pipe and only a little amount of it escapes, resulting in high storage but low discharge. Then, more water flows out of the pipe as the inflow increases higher and the pipe becomes fuller, hence increasing storage further. Finally, the inflow decreases and stops, the storage declines as all of the water in the pipe has been discharged. On the other hand, when the pipe is short, the same process happens, but the discharge can quickly flow out of the pipe without being accumulated inside. Therefore, the discharge varies less.

With the same length, a lower slope results in higher storage. This is expected since when the pipe is flatter, water travels more slowly throughout the pipe and hence gets accumulated more inside the pipe before discharging. This means that the pipe is run fuller with low slope.

The linear slope representing the SQ relationship is also more vertical for lower lengths and much more horizontal, i.e. the slope is smaller, when the transport stretch becomes longer. This translates in the increasing storage coefficient c with increasing pipe length. This phenomenon can be observed in Figure 5.6.

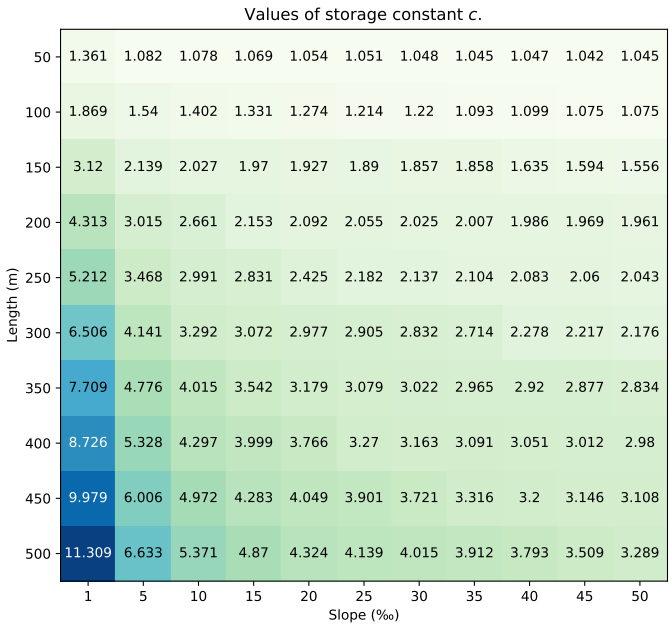

Figure 5.6: Storage constant c (minute)

Figure 5.6 demonstrates the values of storage constant c in the resolution of minute. Once again, an extremely evident trend can be observed, which is flatter and longer pipes always have higher delay constant. As mentioned before, this agrees well with the behavior in Figure 5.5. It is obvious that water takes more time to travel throughout a long pipe compared to a short and steep one.

5.1.2.1 Extrapolation Model to Find Slope of SQ Function Based on Storage Constant c

The derived storage constant c made it possible to develop a model to extrapolate the values of c when the transport stretch's length and slope exceeds the tested range based on the values of c obtained above. This model is built for calculation of the slope of the SQ function $1/c$.

After careful visual analysis of the distribution of slopes of the SQ function given the physical characteristics of the transport stretch, it has been determined that $1/c$ is mostly inversely proportional to the length and logarithmically dependent on the slope (see Figure 5.7). With this knowledge, the following model for the prediction of the slope of the SQ relationship was defined:

$$1/c = \theta_0 + \theta_1 \log(s) + \theta_2 \frac{1}{l}$$

where $1/c$ is the predicted slope of the SQ function, s and l are the slope and the length of the transport stretch respectively and $\theta_0, \theta_1, \theta_2$ are the three coefficients that are found through least squares linear regression. The first coefficient, θ_0, is the intercept, θ_1 and θ_2 instead, indicate how much the logarithm of the slope and the inverse of the length contribute to finding the slope of the SQ function. When taken together, these three parameters define a plane that approximates the values of $1/c$ and tries to predict them outside the known range of lengths and slopes.

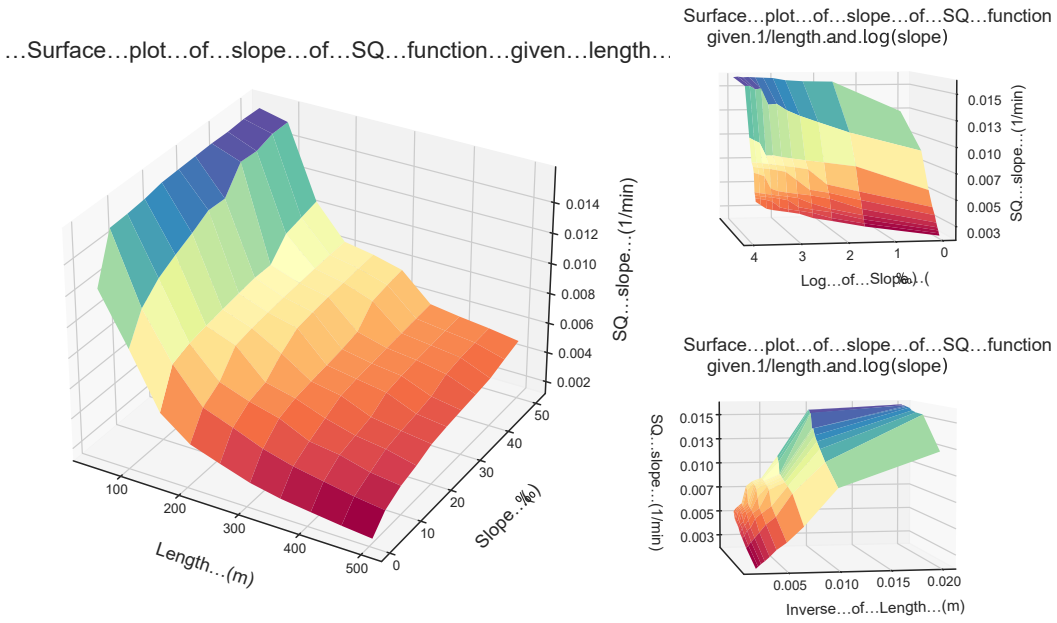

Figure 5.7: Plot of the slope of the SQ function $(1/c)$ given length and slope (left) and given the logarithm of length and the slope (right). The two plots on the right highlight how the slope of the SQ function is mostly linearly dependent on the logarithm of the slope and the inverse of the length

The model, trained through minimization of the root mean square error (least squares regression) is shown in Figure 5.8 and has the following coefficients:

$$1/c = -0.159 + 0.039 \log(s) + 130.662 \frac{1}{l} \tag{5.1}$$

Figure 5.8 shows the fitted and extrapolated model for SQ slope $1/c$ out of tested range for each length and slope combination.

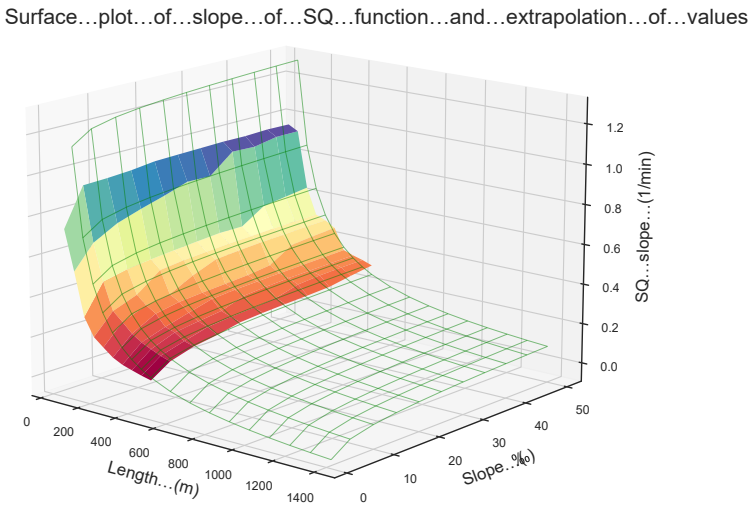

Figure 5.8: Extrapolation of the SQ slope $1/c$ (pictured as a wireframe) outside the range of known values of the SQ slope (pictured as a surface plot)

5.1.3 Discharge Using Linear SQ Relationship

Using the obtained values of c from subsection 5.1.2, the linear relationship between storage and discharge can be drawn and used to describe the actual conceptual model's discharge for the transport stretch.

5.1.3.1 RMSE for SQ Conceptual Model

Figure 5.9 presents the errors obtained from comparing the conceptual model using SQ function to describe the transport stretch with the outflow results achieved from MU+. These errors are multiplied by 10^3 to make it easier to read.

Figure 5.9: RSME between SQ model and MU+ (values multiplied by 10^3) (left) and the difference between Nash model and SQ model (values multiplied by 10^3) (right)

The errors become larger when the pipes become longer and flatter, as expected. Since this SQ model is built based on the results from the Nash model's parameters, these errors also makes sense. In the Nash model, the RMSE of these long and flat pipes are already larger than the shorter and steeper ones (Figure 5.2). Therefore, having a secondary model based on it will have the same large errors where the original model has.

The differences between the errors of the Nash model and the SQ model ($RMSE_{Nash} - RMSE_{SQ}$) are always positive, indicating that the SQ model is consistently more erroneous compared to the Nash model, which is as expected. The SQ model becomes more flawed compared to the Nash model when long pipes are simulated. This links back to Figure 5.5, where for longer pipes, the relationship between storage and discharge is not as linear as for shorter pipes, but rather forms a loop. Therefore, a single linear fit to represent this relationship cannot describe accurately the behavior of the outflow. For shorter pipes and higher slopes, the differences between the two conceptual models are negligible.

5.1.3.2 Discharge Comparison Between SQ Conceptual Model and MU+

Figure 5.10 shows the comparison for the simulated discharges between MU+ and the conceptual models using Nash linear reservoir cascade and using SQ function.

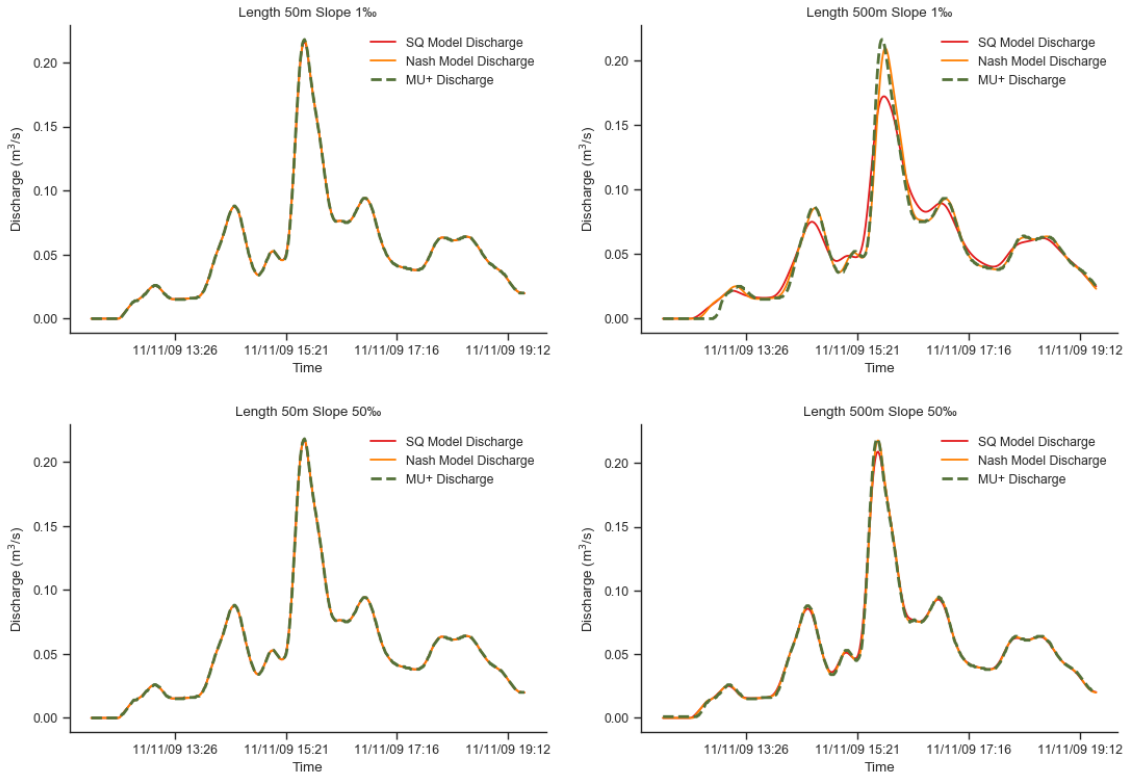

Figure 5.10: Discharge comparison between MU+ and conceptual SQ function and conceptual Nash model

With short length, both conceptual models work very well in simulating the discharges that can match very closely to the results from MU+. The opposite happens to the longer pipes, which corresponds to the errors and the explanation in subsubsection 5.1.3.1. Here, with length 500m, the SQ model could not simulate the high peaks correctly both magnitude-wise and time-wise. This shows that there has been an effect of smoothing out the outflow when applying the SQ function for long pipes. Both the Nash model and linear SQ model cannot capture the delay at the beginning of the rain for length 500m accurately.

5.2 Virtual Cities Simulations

This chapter presents the results of virtual cities simulations, which contains of the outcomes for different length and slope combinations as well as each city's discharge of each scenario. Furthermore, the model evaluation results are also illustrated.

5.2.1 Length and Diameter Choices

Figure 5.11 shows the discharge of the conceptual model with different length and diameter combinations. In this case, only the Star city's coarse compartmentalization level is presented because in this case, the entire city is represented by only one compartment, making the effect of having various combinations clearer. At

the start and end of the rain, i.e. before and after the biggest peak, only the red and orange lines can be seen, representing the longest sum length and average diameter, and the weighted average length and weighted average diameter simulations, respectively. The other two scenarios are identical to these two, therefore are not observable. Apart from the highest peak that can show clearly all 4 scenarios, the longest sum length, max diameter and longest sum length, average diameter are the same values, and the remaining two weighted average length scenarios share the same discharge curve.

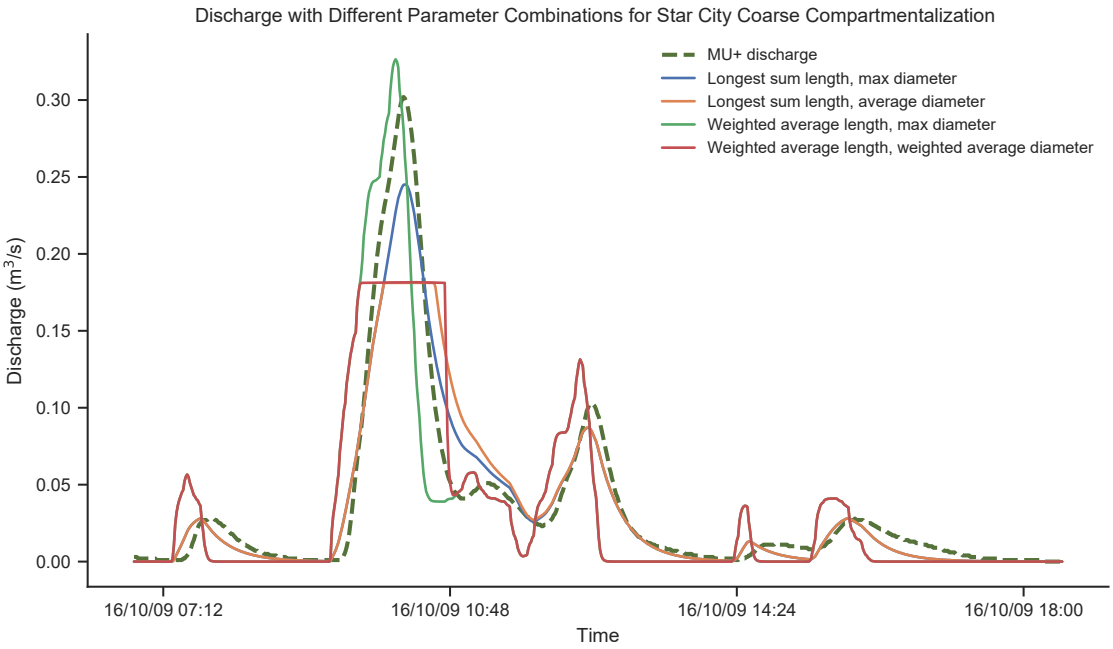

Figure 5.11: Discharge with different parameter combinations of Star City

It is evident from Figure 5.11 that the diameter determines the cutoff point in the discharge of the conceptual model. Regardless of length, when average diameter of all pipes is used, it affects directly the maximum discharge q_{max} of the SQ function. Therefore, the flow is restricted and for big peaks, it would not be able to give accurate results. On the other hand, when using the maximum diameter of all the pipes, the model will always select the biggest diameter, meaning that peaks can be simulated without any restriction, which is similar to having the model without q_{max}. When there is only little rain, both ways of calculating diameter does not affect the discharge since it does not reach the cutoff point.

The selected length affects directly the shape of the peaks in the conceptual model. When using the average length, the entire city is represented with a really short pipe. When the pipe is short, there is nearly no delay. Thus, when inflow arrives in the pipe, it will exit instantly. For this reason, the peaks of the discharge in this case are always higher and narrower, while occurring sooner compared to that of MU+. Conversely, when the length is calculated using the sum of longest transport

stretch, it can recreate the MU+ discharge much more closely. With such long pipe, the flow is more delayed and smoothed out. Thus, the peaks are not as high anymore and the time that the peaks occur is closer to that of MU+.

All values of NSE of length and diameter combinations are presented in the following tables for the two cities with all scenarios (Table 5.1, Table 5.2). The numbers in bold are the best NSE. For the full network scenario, all NSE values are the same because there is no lumping.

Table 5.1: NSE of different parameter combinations for all scenarios of Star City

Length	Longest sum		Weighted average	
Diameter	Maximum	Weighted average	Maximum	Weighted average
Coarse	**0.9626**	0.9295	0.7092	0.7091
Medium	**0.9647**	0.9647	0.7759	0.7759
Fine	**0.9686**	0.9573	0.8055	0.8084
Full network	0.9342	0.9342	0.9342	0.9342

Table 5.2: NSE of different parameter combinations for all scenarios of Strip City

Length	Longest sum		Weighted average	
Diameter	Maximum	Weighted average	Maximum	Weighted average
Coarse	**0.8216**	0.8216	0.7796	0.7841
Medium	**0.9448**	0.9448	0.7992	0.7992
Fine	**0.9497**	0.9497	0.8631	0.8630
Full network	0.9480	0.9480	0.9480	0.9480

It can be seen that the conceptual model simulates best when using the sum of longest transport stretch for the length and maximum diameter of all the pipes in the compartment. The different ways of calculating diameter only have an effect when the discharge is big enough, meaning that when there are big peaks in the discharge, using average diameter would cut off the peak since the pipe has reached it maximum capacity. In the coarse level of compartmentalization, the difference between using maximum or average diameter is the most pronounced. There is only 1 compartment, therefore the many small pipes upstream contribute to decreasing the average diameter, making the pipe capacity decline more while the runoff of the entire city flows into it, hence there are more cutoff points in the discharge. On the other hand, when dividing the cities further, the variation between upstream and downstream pipes is less, which makes the pipe diameter size more reasonable compared to the amount of inflow it has to convey. For this reason, the NSEs do

not differ when using maximum or average diameter, and only a slight increase in NSEs is seen with the maximum diameter used, in the case of using longest sum length.

The parameter that determines most strongly the accuracy of the conceptual model is how the length is computed. Thus, there is a big difference in NSEs between using the average or longest pipe length. The reason is that the CM for transport stretch takes into account only the length and slope of the pipe. Thus, when the length varies, the slope of the SQ function also changes, affecting the delay and shape of the discharge considerably. Weighted average length implies a much shorter length of the transport stretch, hence always results in less delay and sharper peaks in discharge, as described in Figure 5.11, affecting the model accuracy. The NSEs of both Star and Strip cities are consistently the best when the longest sum length and maximum diameter are used. Hence, for further analysis of compartmentalization, only this parameter combination will be applied.

5.2.2 Star City Compartmentalization

5.2.2.1 Discharge Comparison

Figure 5.12 shows the comparison of the discharge of the conceptual model compared to that of MU+, with 4 scenarios of compartmentalization. A clear trend can be observed: the more compartments there are, the higher and narrower the peaks of discharge are in the conceptual model. For the biggest peak where the discharge reaches nearly 0.3 m^3/s, simulating with full network gives the best match both magnitude-wise and time-wise. However, full network simulation results in overshoot in smaller peaks with little delay. Even though the discharge starts at the same time when rain arrives for all scenarios, the delays in peak time vary among cases, with the full network peaks arrives first, follows by the coarse, medium and fine levels.

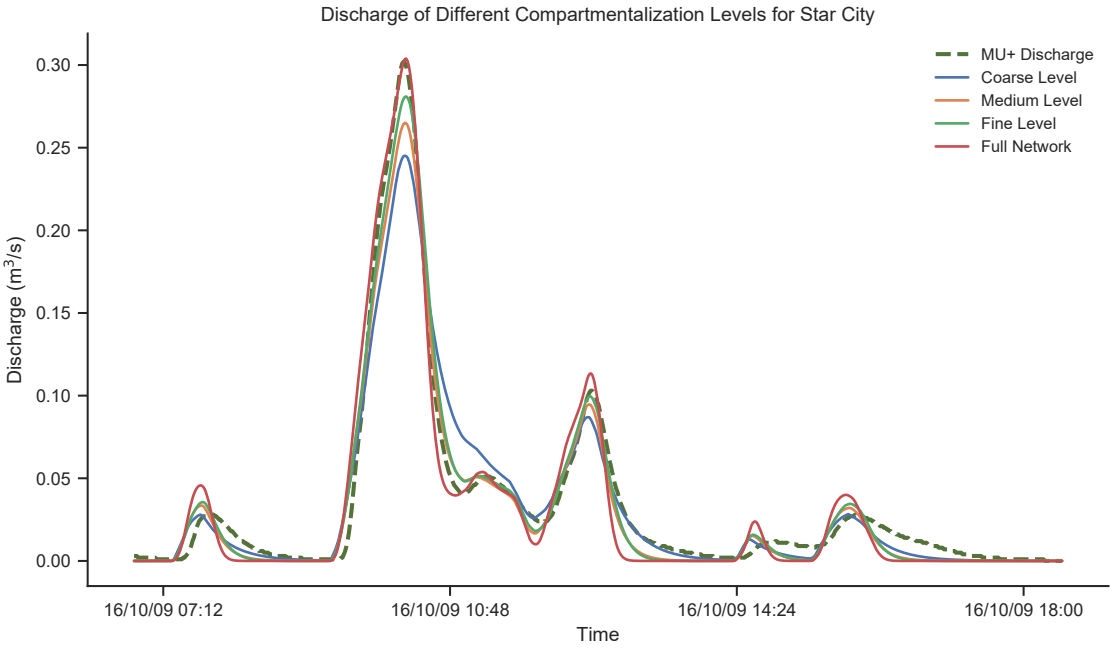

Figure 5.12: Discharge of different compartmentalization levels for Star City

The fact that the full network model delays the least in small peak time goes against our initial expectation, since this case simulates the drainage network exactly as it is, with different inflows entering the system at different time based on the position of the manholes. Furthermore, we expected that this level of compartmentalization would be the least erroneous since each pipe in this scenario is quite short, often between $65 - 100$m, which has the smallest error simulating discharge using SQ function in the case of transport stretch conceptualization. The full network simulation only works well in terms of magnitude and delay when there is enough inflow inside the system. For smaller rain events, it cannot model correctly.

All scenario's discharge starts to rise at the same time, which is earlier than the MU+ discharge. This indicates that the rain starts at that time, and MU+ discharge is delayed. This effect is already pointed out in subsubsection 5.1.3.2, where none of the conceptual model for transport stretch can replicate this delay. Therefore, it is expected that the city simulation cannot reproduce the exact delay at the beginning of the rain.

The remaining three scenarios behaves similar to our expectation, with the finer compartmentalization capturing better the discharge in how smoothed out and delayed the MU+ outflow is. The shape of the peaks is mainly determined by the length used to represent the compartment. The more compartments there are, the shorter the length. With shorter pipes, the peaks become narrower and higher, which in this case matches with the big peaks from MU+ better. Overall, from Figure 5.12, it can be seen that even the coarse compartmentalization can still capture the small peaks well. On average, fine level model seems to produce the closest dis-

charge compared to MU+. However, simulating the full network is the best model to describe high peak flows.

To inspect and directly compare the discharges of MU+ and the conceptual models throughout the entire 3 months period, Figure 5.13 gives this information.

Figure 5.13 confirms the phenomena seen in Figure 5.12, where the full network simulation matches high peaks very well. The finer the compartmentalization, the better match between CM and MU+ with high flows. In general, the CM tends to overestimate the low flow discharges and underestimate high flows. Fine compartmentalization still produces the best fit overall, with much less difference and variation compared to MU+, while coarse compartmentalization performs the worst.

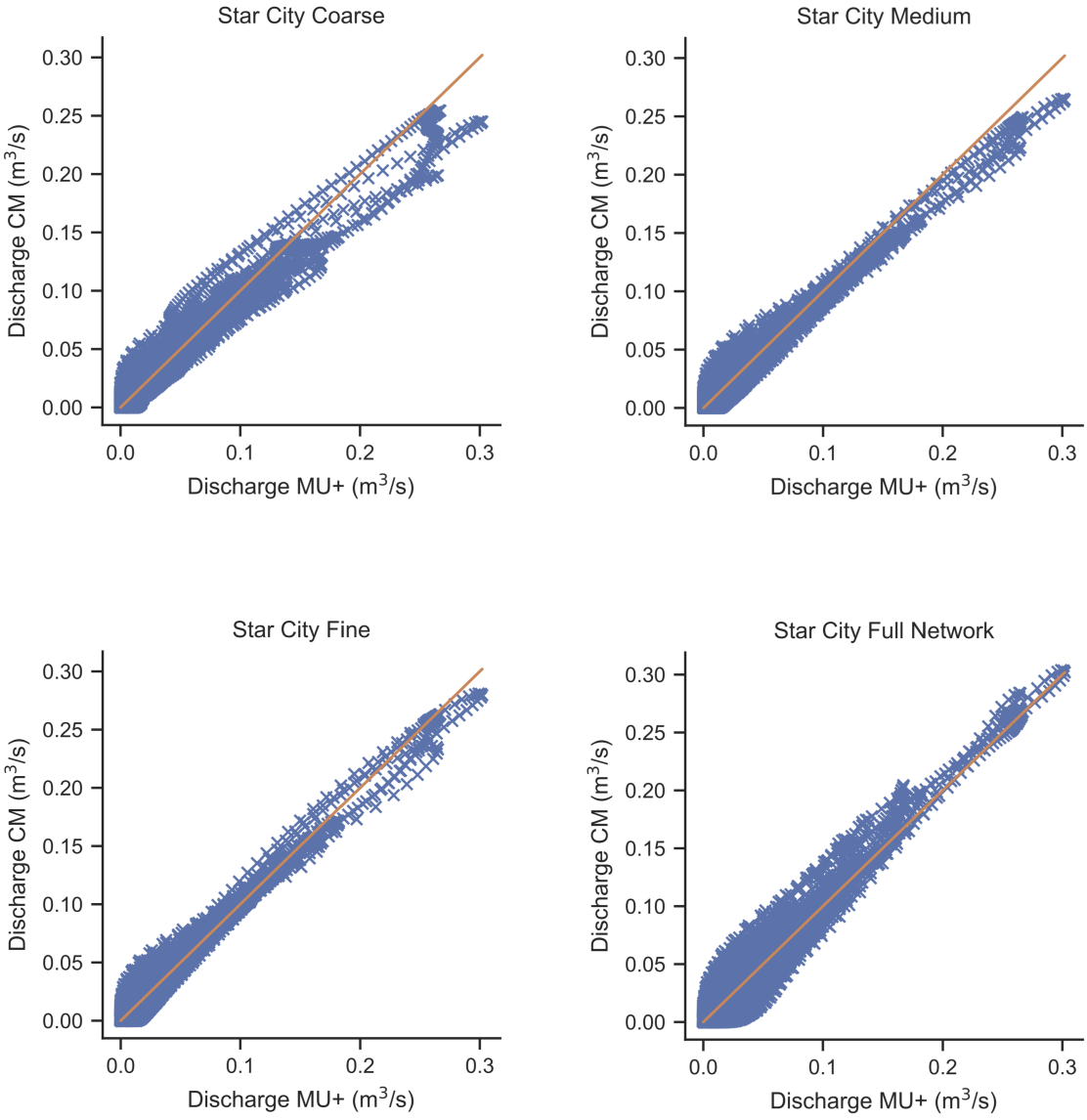

Figure 5.13: Comparison between discharge of conceptual model and MU+ for Star City in 3 months with 4 levels of compartmentalization, orange line shows the perfect fit

5.2.2.2 Conceptual Model Evaluation

To quantitatively evaluate how well the conceptual models perform, NSE, PEP, $PDIFF$ and PTDIFF are presented. The values of NSE for each scenario can be seen in Table 5.1, note that only the longest sum length and maximum diameter are simulated for the city compartmentalization. The NSEs for all scenarios are quite high, most are around 0.96, indicating that the conceptual model works well to predict discharge compared to MU+. Simulation with full network gives the lowest NSE, most likely due to the overestimation of low flows along with not having enough delay. Coarse, medium and fine models have rather similar $NSEs$, with fine model performing the best. The reason for this similarity is due to the relatively small differences among these models when simulating low flow, which is the majority of the cases. The big peaks are the main points of variation among them. Even though the fine scenario performs the best, medium level is sufficiently accurate, since the difference in $NSEs$ between these two models is marginal.

Figure 5.14 indicates the evaluations of the conceptual models when only peaks are considered, since the main aim of the conceptual model is to correctly simulate peaks. Table F.1 presents all mean values of PEP, $PDIFF$, $PTDIFF$ along with their standard errors for Star city.

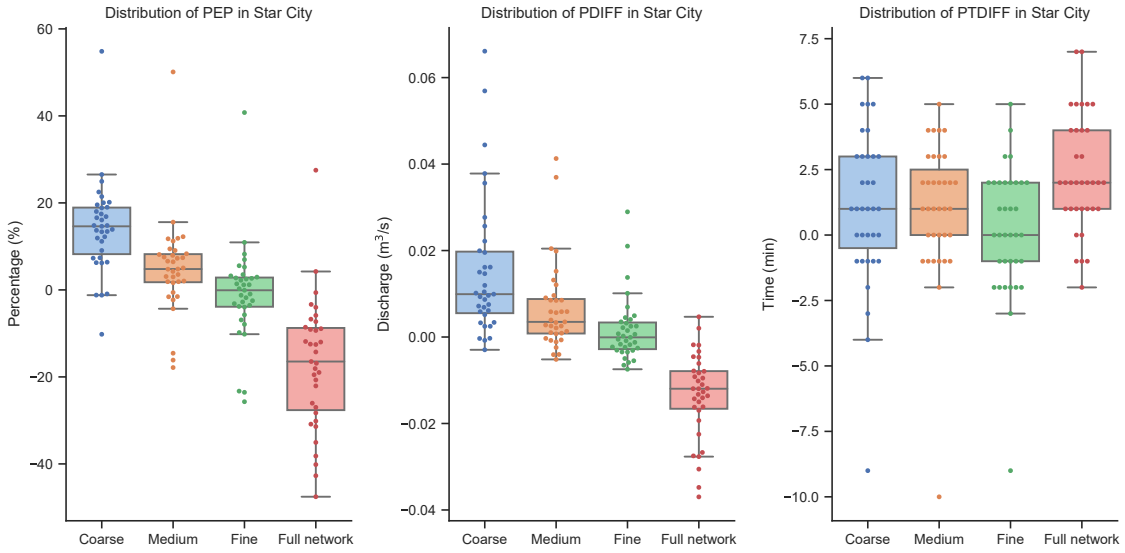

Figure 5.14: Box plots of PEP, $PDIFF$ and $PTDIFF$ of Star City

The PEP values indicates the percentage difference between peaks of MU+ and CM. Full network simulation results in peaks that are often higher than MU+'s peaks, which is why its PEP is negative. This agrees well with the peak behaviors already discussed above. The three remaining scenarios all have CM peaks being lower than MU+ peaks, hence the mainly positive distribution of $PEPs$. This indicates that the conceptual models often underestimate the peaks, which agrees with Figure 5.13. Even though many peaks are small, which is categorized as small flow,

and subsubsection 5.1.3.2 concluded that for these small flows, the CM often overestimate the discharge, these peaks are not taken into account when calculating PEP. The reason is that for each rain event, only the biggest peaks are considered, not the smaller ones. It means that when a rain event consists of several peaks, the smaller ones are neglected and only the highest peak is used for computation. Therefore, the peak evaluation parameters only aim at assessing high flows. This is applied to the calculation of $PDIFF$ and $PTDIFF$ as well, for both Star and Strip cities. Fine compartmentalization performs the best in this case, while full network simulation is once again the worst performing model. Moreover, full network scenario also results in the biggest variation in PEP.

The difference between CM and MU+ discharge, magnitude-wise, can be seen from the $PDIFF$ values. This parameter follows the same trend as PEP, where full network has negative $PDIFF$, implying often higher peak discharges than MU+. In this case, the coarse scenario instead has the biggest variation in $PDIFF$. Fine level of compartmentalization still performs the best with low $PDIFF$ and small variation. Overall, all models do not have too significant peak discharge differences compared to MU+.

$PTDIFF$ indicates the difference in the time that the peaks happen. Most values of $PTDIFF$ indicates that the peaks often arrive earlier in the conceptual models for all scenarios, which is in agreement with Figure 5.12. On average, fine scenario still has the closest resemblance to MU+, but the differences between scenarios are not too significant. In general, all of the conceptual models are still capable of simulating the peak times with low errors.

5.2.3 Strip City Compartmentalization

5.2.3.1 Discharge Comparison

Figure 5.15 describes the discharges obtained from 4 compartmentalization scenarios, in comparison with the discharge computed in MU+. Basically, the same trend happens as in the case of Star city. The finer the compartmentalization, the better the conceptual model simulates the discharge. When using the full network simulation, the biggest peak is captured most accurately, but slightly bigger in magnitude, while the smaller peaks are still experiencing overshoot and narrower peaks.

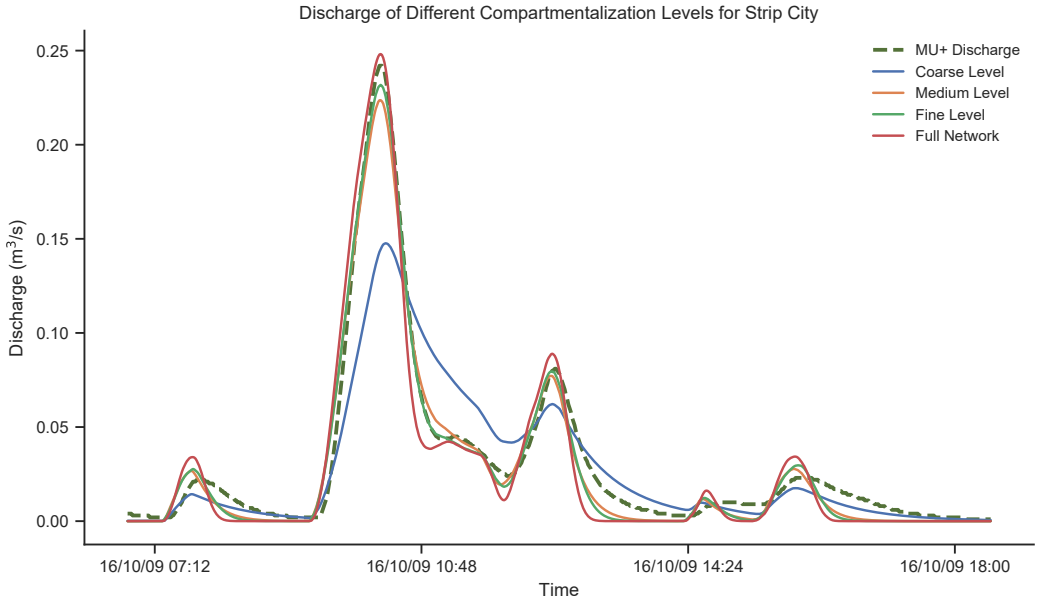

Figure 5.15: Discharge of different compartmentalization levels for Strip City

Coarse scenario for the Strip city performs extremely bad, with all the peaks being significantly lower than MU+, while peak time is too delayed. This is because unlike Star city, Strip city spreads out longer horizontally. This is caused by the length used to derive storage constant being much longer at around 1.4km, when the entire city is lumped into only 1 compartment. The result of having such a long stretch is that the discharge is much more smoothed out, similar to the phenomenon mentioned in Figure 5.10. For the other scenarios, the discharge behavior follows the same pattern and explanation as in Star city.

Figure 5.16 indicates the comparison between MU+ and conceptual models' 3 months long data. Since the coarse model has too long length, its behavior is rather chaotic the higher flow there is. For Strip city, the MU+ discharge itself is already lower than that of the Star city. Medium, fine and full network models all can capture high peaks relatively well. However, for smaller flows, the full network model results in both overestimation and underestimation that varies the most. In this case, fine model once again results in the discharges that are closest to MU+.

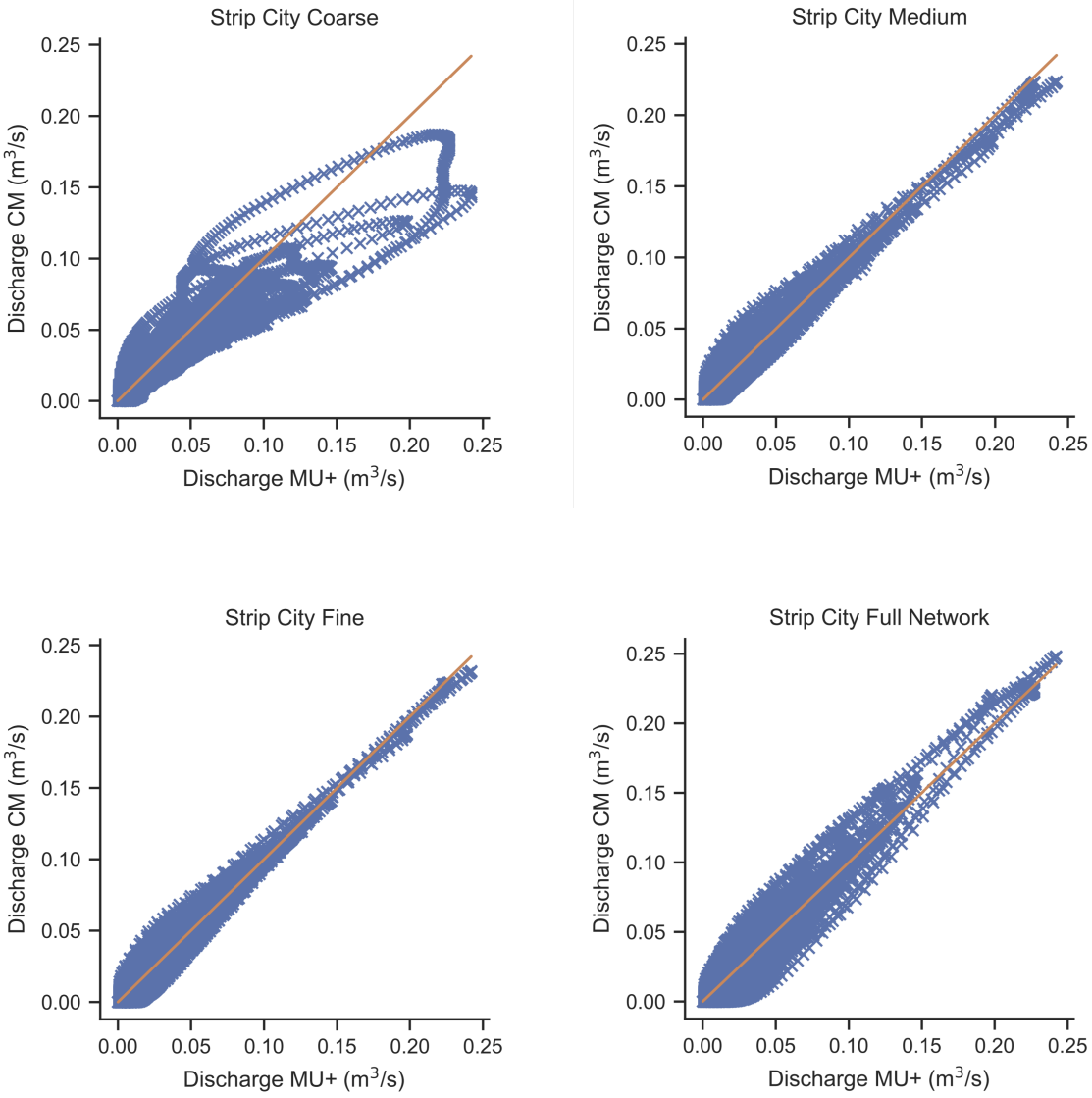

Figure 5.16: Comparison between discharge of conceptual model and MU+ for Strip City in 3 months with 4 levels of compartmentalization, orange line shows the perfect fit

5.2.3.2 Conceptual Model Evaluation

The values of NSE for the Strip city is available in Table 5.2 (first column). Overall, the NSE values also follow the trends of Star city, increasingly good with finer compartmentalization. However, the full network model in this case performs slightly better than the medium level of compartmentalization. The fine model has the best performance, but all NSE values for Strip city are lower than those of Star city.

Figure 5.17 shows the errors regarding peaks simulation between the discharges of conceptual models and MU+. The mean and standard error values of these evaluation are available in Table F.2.

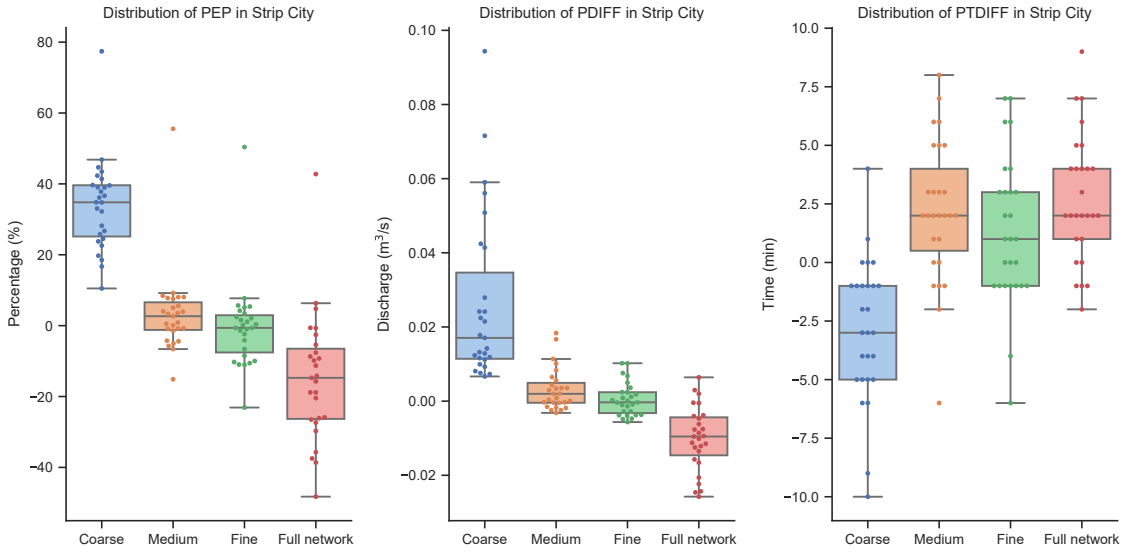

Figure 5.17: Box plots of PEP, $PDIFF$ and $PTDIFF$ of Strip City

Similar to the Star city, $PEPs$ of full network model in Strip city also stays at a negative value, meaning that the peaks of MU+ are smaller than the peaks of conceptual model. At the same time, the variation in $PEPs$ of full network model is still the highest. The fine level of compartmentalization shows a slight negative distribution of $PEPs$ as well. However, except for the extremely smoothed out behavior in the coarse model, which results in a very high PEP, the remaining models perform better than those of Star city, with $PEPs$ closer to 0%. The fine model is still the best performing one of all.

$PDIFF$ shows a similar behavior as Star city. Nevertheless, the differences in magnitude of peaks are even smaller for Strip city, with less variation as well. While the fine model results in $PDIFF$ closest to 0, all models can be said to have a very small difference compared to MU+.

The $PTDIFF$ values for coarse model are often negative, indicating that this model over-delays the peak time compared to MU+. On the other hand, all remaining models experience early peak discharges. Compared to Star city, the conceptual models of Strip city performs slightly worse in terms of $PTDIFF$.

5.2.4 Computational Time

The main advantage of having a conceptual model is how fast it can simulate the urban drainage network and produce results. Table 5.3 shows the time it takes to simulate the drainage network in MU+ as well as by using conceptual models. The time shown here is only for computing the conceptual models using longest sum length and maximum diameter.

Table 5.3: Computational time between MU+ and different levels of conceptual models for Star and Strip cities

	Star City	**Strip City**
MU+	7 mins 26s (446s)	8 mins 17s (497s)
Coarse Level	9.22s	7.22s
Medium Level	9.61s	6.98s
Fine Level	9.93s	12.14s
Full Network	24.75s	21.91s

It can be seen that the conceptual model reduces computational time significantly, being at least 20 times faster for the full network simulation. In case of coarse, medium and fine levels, the conceptual models are around 50 times faster for the Star city and 70 times faster for Strip city. The time it takes to compute the conceptual model is obviously dependent on the number of compartments, increasing with more compartments included. However, even when every single pipe in the network is simulated, this process still takes less than $30s$, which is extremely fast. This allows very versatile use of the conceptual models, and with such high speed, it can be used for planning or even real time control. Additionally, the code script written for this research is also automated enough to allow flexibility in creating compartments and connecting them together. Therefore, the time required to set up the entire network, i.e. stating which runoff flows into which pipe, how many catchments are included in a compartment and how each compartment flows into another one, etc., is actually not different between different levels of compartmentalization. It means that once the input is available, the model setup time is not dependent on how many compartments there are. However, it should also be taken into account that the surface runoff of each catchment is taken as a result of MU+ model. Had it been that the conceptual models also produces surface runoff results by itself, it could have taken a little more time.

5.3 Test with Increased Initial Storage SQ Function

It was noticed that with longer pipes, when inflow arrives in the system, the delay is not linear and it would take long for discharge to start coming out of the system, as explained in Figure 5.3 and can also be seen in Figure 5.10. The delay in discharge at the beginning of the rain cannot be simulated even with the optimized Nash model, and hence also the SQ model. This phenomenon is also briefly mentioned in subsubsection 5.2.2.1, that regardless of the scenario, all conceptual models start to discharge at the same time, which is earlier than the MU+ discharge, when rain first falls down. This results in the motivation to optimize another SQ model with 2 pieces of linear functions for the transport stretch. This new model manually forces the system to accumulate enough storage before discharging. It means that

this model adds an extra delay at the start of the rain event. In this model, extra initial storage S_0 is selected, so that when the storage in the system increases from 0 to S_0, the system will only discharge a q_0 of $0.01m^3/s$. When the storage is bigger than S_0, the discharge will follow an optimized SQ slope, see Figure 5.18.

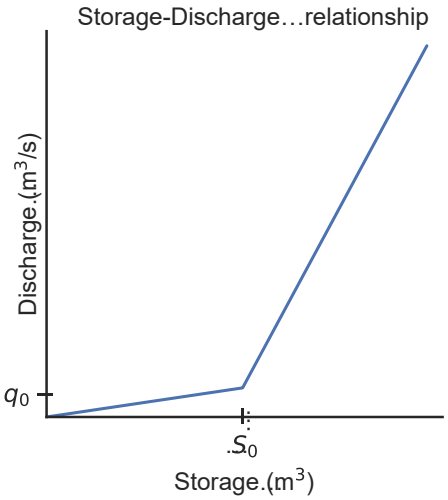

Figure 5.18: Example of piecewise SQ function using increased initial storage

The process and results of optimizing this new piecewise SQ function is described in Appendix G. The process also starts with optimizing the piecewise SQ function for transport stretch, and then the optimized results are used to simulate the virtual cities. However, this optimization does not go through the Nash model.

It was concluded that using this piecewise SQ function with increased initial storage can solve the issue of having too little delay at the beginning of the rain. However, this model creates a bigger problem after peaks have been reached, which is that the pipe empties too slowly. When simulating the cities, the piecewise SQ function does not change the magnitude of discharge for the big peaks compared to when using single linear SQ functions. Nevertheless, at the beginning of the rain, the discharge of the first peak does delay more but its magnitude becomes too small compared to MU+.

6 Discussion

This section discusses the evaluation of transport stretch conceptualization results as well as the effect of compartmentalization on the accuracy of the conceptual models. Additionally, some model limitations are considered along with the recommendations for future work.

6.1 Transport Stretch Conceptualization

The transport stretch conceptualization's results are discussed in this section. Additionally, the accuracy of the simulated storage coefficient is evaluated and the relationship between the Nash model's lag time and c is explained.

6.1.1 Results of Transport Stretch Conceptualization

The optimization process of the conceptual Nash model allows finding highly accurate parameters to describe the transport stretch. This leads to the fact that the resulting discharge is significantly close to the discharge from MU+. Even though the conceptual Nash model has more difficulties simulating the dynamic and non-linear discharge behavior for longer transport stretch, it can be said that the Nash model still has relatively high accuracy with remarkably low errors.

It is also known that the process of optimization did exclude dry periods, which makes the optimization process ignore the last parts of the recession periods of the discharge. However, this exclusion is not significant enough to change the obtained parameters, since the recession periods not considered are not too long. Meanwhile, the model's main aim is to simulate correctly the delay and the peak discharges, which are captured precisely by the selected periods already.

Building a secondary model based on the results of the Nash model makes the conceptual discharge more erroneous compared to that of MU+, but once again, the error is not too significant. The most extreme inaccuracy occurs at the extreme slope of $1‰$ due to the discharge hysteresis effect. Other than that, a linear approximation of the SQ relationship should be enough for the tested lengths and slopes since the function is often linear (Appendix E).

With the lower errors achieved from using the Nash model, one would argue that using the k and n parameters to predict discharge would make a more accurate model. Theoretically, the conceptual model using a single linear SQ function is the same as using the Nash model with $n = 1$. It is possible to simulate the SQ model using multiple SQ functions in series, with each SQ function acts as 1 reservoir, which essentially means that the Nash model can be applied to the SQ model to recreate the non-linear effects in flat long pipes. However, doing so would increase

the computational time, which counters the main aim of a conceptual model, since as can be seen in Figure 5.1, some values of n reache 300. Another method would be to have a fixed number of SQ functions in series (such as KOSIM model) and optimize the value of k, so as to have the discharge hysteresis taken into account without compromising the computational time, since simulating with $n = 300$ can be unnecessarily detailed. Additionally, it is crucial to remember that k and n only work in combination with each other. Otherwise, the obtained k and n do not follow a pattern and it would be impossible to derive them separately. Therefore, it can be said that a linear SQ function is adequate for simulating the water flow for a single transport stretch.

6.1.2 Accuracy of Simulated Storage Constant

The storage constant c obtained through using the Nash model is only more erroneous when simulating discharge due to the fact that it did only a linear approximation and cannot describe the discharge hysteresis. However, the method to derive this SQ function from the Nash model's k and n still gives accurate results. This is proven by optimizing the linear storage constant c through *surrogate model engine for water networks by DTU* directly without using the Nash model (Appendix H).

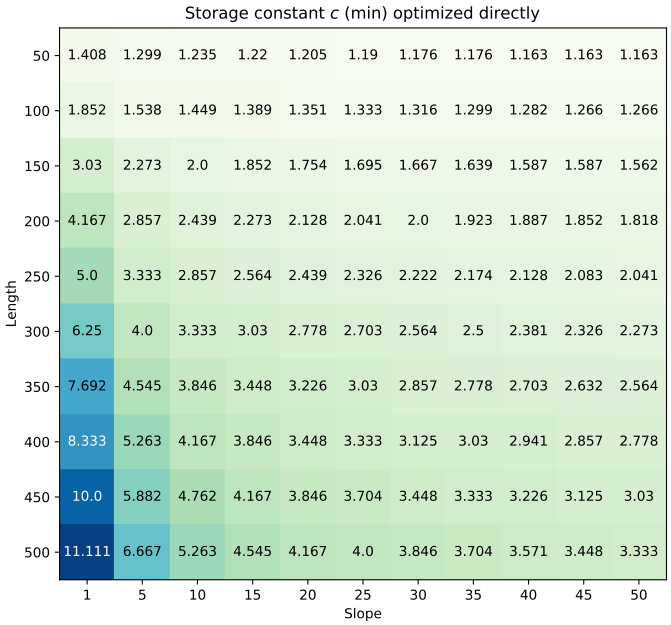

Figure 6.1: Storage constant c (minute) optimized directly not through Nash model

Figure 6.1 shows the storage constant c derived directly through optimization and not using the Nash model as an intermediate step. Compared to Figure 5.6, these values are almost identical. This is only to show that the storage constant c derived from the Nash model is very accurate and the optimization steps taken to obtain these values are correct. As mentioned in section 4.1, it is more convenient to optimize c directly. However, due to the easier implementation of the Nash model

at the beginning of this study, all results are based on the storage constant derived by Nash model. Moreover, when simulating the reservoir cascade with the results from the Nash model, the discharge hysteresis effect can be seen clearly (Figure 5.5), making it easier to understand and explain the system behavior compared to when having only an optimized SQ slope. The direct optimization of c was only computed as a validation to prove that all calculations to get c in this research are correct and hence did not get applied further to simulate the virtual cities.

6.1.3 Lag Time of Nash IUH and Storage Constant

Theoretically, the concept of lag time t_{lag} for the Nash model (computed with kn) is the same as the lag time, or storage constant c of the SQ model. It is possible to observe this fact from Figure 5.2 and Figure 5.6. The differences between them are only marginal, with maximum difference in pipe of length 500m and slope $1‰$ due to the discharge hysteresis. The fact that these two parameters are extremely close to each other proves that the conceptual model to find the storage constant c works accurately. It also explains why when k and n are obtained, simulating the differential equation system to find storage and discharge did not significantly change the value of storage constant c when a random rain is chosen, not the rain used to optimize Nash model's parameters (Appendix B). This means that the relationship between storage constant c with k and n remains the same regardless of the rain used to find the c. Additionally, this provides another way of calculating the storage constant c when knowing the values of k and n without the need to simulate through the differential equation cascade, which is that $c = kn$. This relationship was not used directly in this study because the fact that c and t_{lag} are the same concepts was not discovered until the results of both were computed.

The reason why t_{lag} of Nash model and c are nearly similar but not exactly the same is because of the way the conceptual model is formulated. The storage constant c in this case is found only through fitting a line through the reservoir cascade's storage and discharge pairs. This makes the fitted line, i.e. storage constant c slightly varies from the t_{lag} especially when hysteresis makes fitting SQ slope more inaccurate.

6.2 Effect of Compartmentalization on the Accuracy of Conceptual Model

For both the Star and Strip cities, a clear trend can be seen, in which the model would perform better when the cities are divided into more compartments, except for the full network simulations. Even though the fine level of compartmentalization performs the best in all evaluations, the full network actually can simulate high peaks perfectly. Therefore, the level of compartmentalization depends on the modeler's aim for the conceptual model. If the aim is simply to have a model that behaves well when having both small and big peaks, then the medium scenario would be sufficient. In this case, it is recommended that each compartment should have an area of $7 - 15$ha. On the other hand, if the main objective is to model big

peaks, then the full network simulation is a better option, without increasing the modeling time significantly.

The fact that even with modeling the full network, only big peaks are accurate shows that perhaps 1 linear SQ function is not enough, since small and big inflow can make the drainage system behave differently. The process is not simply linear as when only 1 pipe is considered in the transport stretch optimization. When there is high storage inside the network, the model can capture very accurately, but the pipes often empty too fast when only a small amount of storage is available inside the system in the conceptual model. When optimizing the transport stretch, the same rain is used to simulate the cities. However, all the surface runoff from this rain is directed only into 1 pipe instead of being divided into many different pipes like for the cities. Therefore, during this optimization, the transport stretch receives a lot of inflow and hence has high storage. Therefore, when translating it into simulating the cities with the full network, the model performs only the best when high amount of runoff enters the system.

The accuracy of the conceptual models does not depend significantly on the configurations of the city, except when the pipe stretch becomes too long as in the case of Strip city. Additionally, the fact that the conceptual model can be applied to 2 cities means that it is flexible and does not depend on anything other than the physical characteristics of the pipes.

Overall, the conceptual models can be said to be sufficiently accurate to simulate the flow inside a drainage system with significantly low computational time required. Moreover, the implementation of it in Python is flexible and automated enough to save time when testing many scenarios. This allows the complete independence of using PDMs for parameters calibration. The optimized parameters for transport stretch now allow modelers to derive the SQ relationship directly whenever the pipe's length and slope are known. Moreover, it also needs to be considered that when simulating CSO, the accuracy of the simulated flow is not the main focus. However, when surcharge is modeled, having a CM that can capture accurately the discharge is important.

6.3 Model Limitations

The single linear SQ function still has problem modeling discharge of long length due to the hysteresis effect, and the problem could be worse when pipes longer than 500m as tested in this research are used, i.e. the discharge would be much more smoothed out. This can be seen the best with the coarse scenario in the Strip city when the length modeled reaches 1.4km. It is expected that the discharge hysteresis effect is even more pronounced with bigger loops that the linear model cannot capture. Therefore, the conceptual model to simulate the city only can perform accurately when the length used to model is short enough of less than 500m. Otherwise, a loop form of SQ relationship has to be used, which is not a

function anymore and becomes more complicated as it is not akin to the linear ones that this research tries to optimize.

Another issue is that when modeling different lengths and slopes scenarios for a single transport stretch in MU+, the model was run with a fixed time step of 1 minute. This is still a coarse resolution when it comes to hydrology, which can affect the accuracy of the conceptual model. However, since there are 110 simulations to run, while MU+ has long computational time, this study is limited to only this coarse resolution. The large time step affects mostly the pipes with fast discharge, i.e. short and steep.

From the results of the parameter selection to model the city, i.e. how to calculate length and diameter (subsection 5.2.1), it is clear that choosing the maximum diameter gives the best results. This is true considering that this study only investigates the movement of water inside the drainage system that does not run with full capacity. Moreover, using the maximum diameter in the calculation is theoretically the same as not establishing any flow limit. It only works when the pipes do not run full or any overflow exists. In cases where CSO or weir discharge or surface spill are modeled, using the maximum diameter would produce wrong results, as the flows are not limited anymore. Therefore, another calculation scheme has to be applied when these cases are being modeled.

Moreover, the model to extrapolate the slope of the SQ function $1/c$ when the length reaches over the tested range of 500m can highly affect the result of the simulation for pipes that are out of the tested range. The exact fit to extrapolate $1/c$ is difficult and can be inaccurate since there is no data for validation. At the same time, with lengths within the range tested, the SQ slope is only obtained by rounding up the lengths to the nearest length tested. This results in some pipes of $65 - 72$m, which are common in the full network simulations, be rounded down to the SQ slope for pipes with 50m length. This can affect the delay and hence the performance of the conceptual model.

For the simulations of the cities, both cities' areas are relatively small compared to an actual city in reality. Additionally, they are of more or less the same size. The model was not tested with bigger sized cities (such as in Thrysøe, Arnbjerg-Nielsen, and Morten Borup (2019), Kroll et al. (2017), and Ledergerber et al. (2019)) to evaluate if the performance is still as good. The formulation of small cities like these results in all pipes having more or less the same slope of $10‰$ because this is the minimum slope required for when dimensioning small flows. Therefore, the slopes being simulated for the cities do not change regardless of scenario or city. This results in the inability to evaluate whether the conceptual model can work with a variety of network slopes, or how the slopes should be calculated, i.e. such as maximum or average slope, to select the correct SQ function for the cities, as with the case of testing different length calculations.

The conceptual model still uses MU+ surface runoff data of each catchment as an

input to simulate the discharge. This eliminates another source of error for the CM while making the CM still dependent on a PDM for this part. It is expected that when the runoff is calculated by conceptual models, the city simulations would become more erroneous.

The evaluation of the model when considering peaks is not entirely accurate, since not all peaks in a rain event are taken into account, but only the biggest peaks due to the difficulties in matching and identifying correctly every single rain event. This results in a limited understanding of how different peaks are simulated by the conceptual model.

Overall, this conceptual model performs very well when modeling pipes that do not run full. However, it cannot model other effects in a drainage system, such as backwater effect, pipes with pressure or pumping, etc. For this reason, the model is more or less a proof of concept, and not yet able to be applied in real life.

6.4 Recommendations for Future Work

From the results and discussion about this model, it is recommended that the optimization of the transport stretch should be extended to longer lengths of at least 2km to have enough data so that the extrapolation model works more accurately. In this case, the direct optimization of the SQ function would be more accurate than deriving it through the Nash model.

Moreover, as mentioned above, simulations with higher resolution of time would make the model much more accurate in capturing the delay. Instead of simulating every minute, it is better to simulate the MU+ with a time step of only seconds. However, the conceptual model run time will increase linearly with the number of steps that are used in the simulation. Furthermore, simulations of bigger cities would provide better understanding of whether the model can work flexibly regardless of city size or not, and the pipes should have various slopes as well. Besides, bigger rain could also be applied to examine the model behavior when the pipes upstream are full. Other than that, this model can also be tested using a real city, not a virtual one anymore.

The simulation of the transport stretch can also be optimized using a fixed number of n. Then, the optimization is only carried out for k for each length and slope combination. This is the same as having storage constant c in a cascade of n numbers of SQ functions in series.

Furthermore, to entirely eliminate the dependence on the PDM, runoff generation should be added into the conceptual models. When the dimensions of the cities are available, information such as catchment area or concentration time, etc. can easily be found. Such information allows straightforward computation of surface runoff from rainfall. The conceptual model can calculate runoff entering each compartment by means of time-area method, similar to what is being used in MU+,

or even directly from the Rational method.

To further develop the conceptual models' usage, other structures of the drainage network can be included. Using the same principle of optimizing the SQ functions, many other network elements such as CSO, basin, surface surcharge or weir discharge can be found. The inclusion of these structures into the conceptual model would allow better completeness of the conceptual models, making it more feasible to be applied independently in reality.

Finally, from the results of running the cities with full network simulation, it can be seen that the big and small rain events behave differently. This suggests that another SQ function for smaller inflow should be added. In fact, the more pieces of SQ functions there are, the more accurate the conceptual models will become. A preliminary test when using the smaller slope of SQ function is shown in Figure 6.2, where the city is simulated with SQ slopes that are $1/2$ and $1/3$ of the currently used SQ slope.

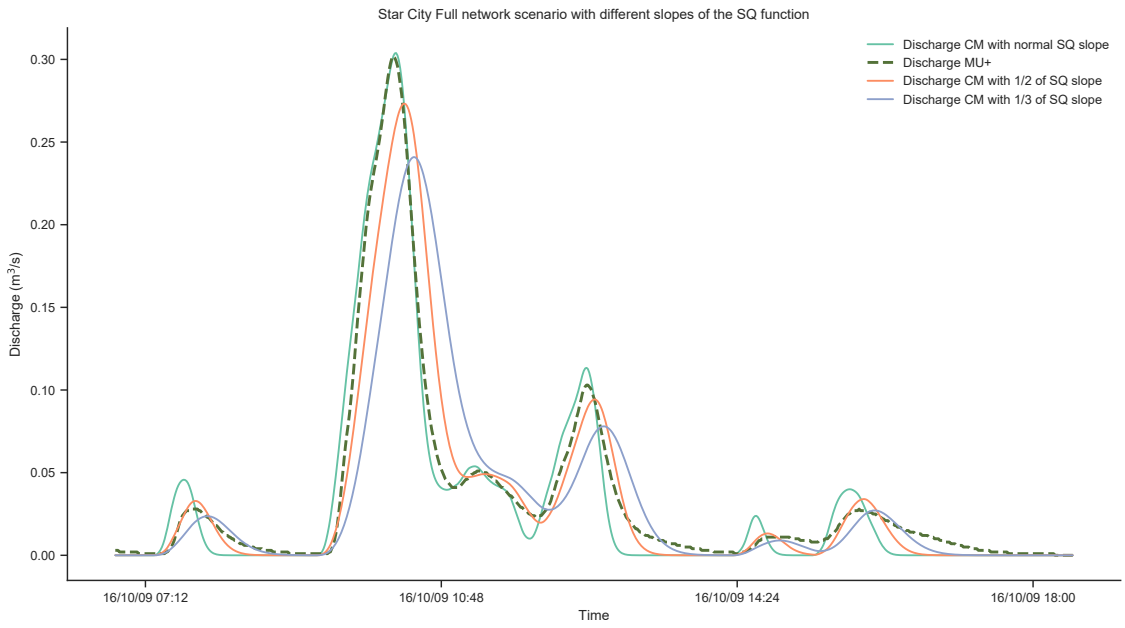

Figure 6.2: Star city full network scenario when run with SQ slope of $1/2$ and $1/3$ of the current slope

This preliminary test shows that different magnitude of rain requires a different SQ function. Bigger rain event would need a bigger SQ slope and for smaller ones, the SQ slope should be smaller. It can be seen that with smaller slopes, even though the biggest peak cannot be captured, the delay and magnitude of smaller peaks improve much better. Therefore, it is recommended that an optimization of the piecewise linear SQ function containing several different slopes to describe various magnitude of rain should be performed to improve the conceptual models.

7 Conclusion

This study addresses the issue where conceptual models are always dependent on other PDMs for calibration of its parameters. A conceptual model to optimize the behavior of a transport stretch based on its physical characteristics (length and slope) is established, and later on, used to simulate the discharge of two virtual cities to study the effect of compartmentalization on the accuracy of the results.

This study has the following main conclusions:

- The conceptualization of a transport stretch can be obtained based on its physical characteristics, i.e. length and slope by using the SQ relationship.
- The linear SQ function (storage coefficient c) performs well with low errors in simulating discharge of a transport stretch, but cannot accurately capture the discharge hysteresis effect in long and flat pipes.
- Transport stretches inside a compartment are best characterized by the longest continuous connected length and maximum diameter of all pipes inside the compartment.
- Conceptual models can simulate virtual cities' discharges accurately with NSE above 0.94 and low peak errors.
- Fine level of compartmentalization gives the best CM discharge compared to MU+ overall, but the full network model simulates the high peaks the most accurate, however fails to capture smaller rain events.
- It is recommended that the medium level (each compartment of $7-15$ha) is sufficiently precise while saving computational time.
- Different shapes of the cities do not affect the quality of the conceptual models. The parameter determining its behavior is the length of the transport stretch.
- Computational time is reduced up to 70 times when using conceptual models compared to when simulating with MU+.
- The SQ function with extended initial storage can delay the discharge at the beginning of the rain, but fails to empty the pipes after rain and therefore is not feasible.
- Limitations exist such as not having enough data when the transport stretch reaches further than 500m, and many other structures of the drainage network are not considered.

- The conceptual models still need to be evaluated in other conditions (such as with bigger cities, bigger rain, etc.) and optimization for piecewise linear SQ function is a promising option to increase the accuracy of the model.

Overall, the conceptual models designed successfully eliminates the dependency on other PDMs. It is proven that the conceptual models are capable of modeling the urban drainage system with results of high accuracy compared to those of MU+ while using only the physical characteristics of the network. This study's conceptual models show great potential for fast and accurate flow simulations for urban drainage systems.

Bibliography

Achleitner, Stefan, Michael Möderl, and Wolfgang Rauch (2007). "CITY DRAIN © - An open source approach for simulation of integrated urban drainage systems". In: *Environmental Modelling & Software* 22, pp. 1184–1195.

Allen, Simon et al. (2012). *Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation. Special Report of Working Groups I and II of the Intergovernmental Panel on Climate Change.*

Borup, M. (2018). "DtuSmModels Free software under the GNU General Public License v3 license, Technical University of Denmark". In: URL: https://github.com/Moboha/DtuSmModels.

Borup, Morten (2019). *Dimensioning the pipe system by hand calculation (Rational Method).*

Butler, D. et al. (2018). *Urban Drainage, Fourth Edition.* CRC Press.

Devia, Gayathri K., B.P. Ganasri, and G.S. Dwarakish (2015). "A Review on Hydrological Models". In: *Aquatic Procedia.*

DHI (2020). *DHI Simulation Engine for 1D river and urban modelling - Reference Manual.*

García, L. et al. (2015). "Modeling and real-time control of urban drainage systems: A review". In: *Advances in Water Resources.*

Gomez, V. M. F. (2011). "Modelling and conceptualization of hydrology and river hydraulics in flood conditions, for Belgian and Bolivian basins".

Hirabayashi, Yukiko et al. (2013). *Global flood risk under climate change.*

Jia, Ning et al. (2019). "Effects of Urban Forms on Separate Drainage Systems: A Virtual City Perspective". In: *Water* 11.4.

Kamradt, B. (2008). *Comparison of sewer modelling approaches within IUWS modelling.*

Kimura, T (1961). "The Flood Runoff Analysis Method by The Storage Function Model". In: *The Public Works Research Institute Ministry of Construction.*

Knight, Donald W and Asaad Y Shamseldin (2006). *River basin modelling for flood risk mitigation.* Taylor & Francis.

Kroll, S. et al. (2017). “Semi-automated buildup and calibration of conceptual sewer models”. In: *Environmental Modelling & Software*.

Ledergerber, Julia M. et al. (2019). “An Efficient and Structured Procedure to Develop Conceptual Catchment and Sewer Models from Their Detailed Counterparts”. In: *MDPI*.

Lei, Tianyu (2020). “Models for fast simulating long term optimization of urban drainage system under different physical characteristics”.

Li, C. et al. (2008). “Use of Nash’s IUH and DEMs to identify the parameters of an unequal-reservoir cascade IUH model”. In: *Hydrological Processes* 22, pp. 4073–4082.

Löwe, R. (2020a). *Example_PipeFlowModel.py*.

– (2020b). *SimulateLinearReservoir.py*.

Meert, Pieter et al. (2014). *Development of conceptual models for an integrated catchment management: Subreport 1. Literature review of conceptual modelstructures*.

Mohan, S. and D. P. Vijayalakshmi (2008). “Estimation of Nash’s IUH parameters using stochastic search algorithms”. In: *Hydrological Processes* 22.17, pp. 3507–3522.

Nash, J.E (1957). “The form of the instantaneous unit hydrograph”. In: *International Association of Hydrological Sciences General Assembly* 45.

Pedersen, J.T, J.C Peters, and O.J Helweg (1980). “Hydrographs by single linear reservoir model”. In: *Journal of Hydraulics Division* 106, pp. 837–852.

Prasad, R. (1967). “A nonlinear hydrologic system response model”. In: *Proceedings of ASCE 93*, pp. 201–249.

Sarma, P.B.S., J.W. Delleur, and A.R. Rao (1973). “Comparison of rainfall-runoff models for urban areas”. In: *Journal of Hydrology* 18.3-4, pp. 329–347.

Sherman, L.K (1932). “Stream flow from rainfall by the unit hydrograph method”. In: *Engineering News Record* 108.

Singh, P.k., S.k. Mishra, and M.k. Jain (2014). “A review of the synthetic unit hydrograph: from the empirical UH to advanced geomorphological methods”. In: *Hydrological Sciences Journal* 59.2, pp. 239–261.

Singh, V.P (1992). *Elementary Hydrology*. Prentice-Hall of India Private Limited.

Solvi, AM et al. (Jan. 2005). "Integrated urban catchment modelling for a sewer-treatment-river system". In: *10th International Conference on Urban Drainage, Copenhagen/Denmark*, pp. 21–26.

Spildevandskomiteen, IDA (2005). *Funktionspraksis for afløbssystemer under regn*. URL: https://ida.dk/media/2992/skrift27funktionspraksisforafloebssystemerunderregn.pdf.

Thrysøe, Cecilie, Karsten Arnbjerg-Nielsen, and Morten Borup (2019). "Identifying fit-for-purpose lumped surrogate models for large urban drainage systems using GLUE". In: *Journal of Hydrology* 568, pp. 517–533.

UN (2018). "68% of the world population projected to live in urban areas by 2050, says UN". In: *UN Department of Economic and Social Affairs*. URL: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html.

Vaes, G. and J. Berlamon (1997). "Characterization of sewer systems with storage / throughflow-relationships". In: *WIT Transactions on Ecology and the Environment* 19.

Vanrolleghem, P.A. et al. (2009). "Making the best of two hydrological flow routing models: Nonlinear outflow-volume relationships and backwater effects model". In: *Proceedings of the 8th International Conference on Urban Drainage Modelling*, pp. 7–11.

Ven Chow David Maidment, Larry Mays (1988). *Applied Hydrology*. First edition. McGraw-Hill Science/Engineering/Math.

VICAIRE (n.d.). *Transfer Function*. URL: https://echo2.epfl.ch/VICAIRE/mod_1b/chapt_4/main.htm.

Wagener, Thorsten, Hoshin V Gupta, and Howard Wheater (2008). *Rainfall-Runoff modelling in gauged and ungauged catchments*. Imperial College Press.

Wolfs, Vincent, Mauricio Florencio Villazon, and Patrick Willems (2013). "Development of a semi-automated model identification and calibration tool for conceptual modelling of sewer systems". In: *Water Science and Technology*.

Wu, Shiang-Jen, Lin-Fang Ho, and Jinn-Chuang Yang (2011). "Application of modified nonlinear storage function on runoff estimation". In: *Journal of Hydro-environment Research*.

A Unit Hydrograph

The unit hydrograph was first introduced by Sherman 1932 and became one of the most popular concepts in hydrology as well as in flood forecasting (P. Singh, Mishra, and Jain 2014). It is a deterministic lumped unsteady flow model often used in CM, which means that the model is spatially averaged without considering randomness and the flow rate is unsteady (Ven Chow 1988). The unit hydrograph is the hydrograph of the catchment's discharge given an effective rainfall with unit depth and duration of time D. The schematic rainfall-runoff process using unit hydrograph is present in Figure A.1.

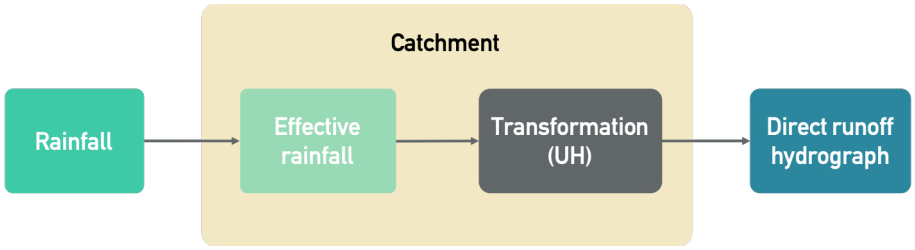

Figure A.1: Presentation of rainfall-runoff process using UH

A.1 Assumptions and Characteristics of Unit Hydrograph

The unit hydrograph model uses a transfer function to derive the discharge hydrograph from any excess rainfall under the following assumptions (Ven Chow 1988; VICAIRE n.d.):

1. The effective rainfall has to spread evenly and uniformly over the catchment.
2. The intensity of such rain has to be constant along duration D.
3. The unit hydrograph of a specific catchment demonstrates the physical characteristics of such catchment, which is unchanged with time.
4. Since the unit hydrograph is not dependent on time, its duration is assumed to be constant regardless of intensity of rain.
5. There is a linear relationship between the unit hydrograph and the effective rainfall.

Concerning assumption 4, it can be seen from Figure A.2 that regardless of rain intensity, the duration of the unit hydrograph still remains the same.

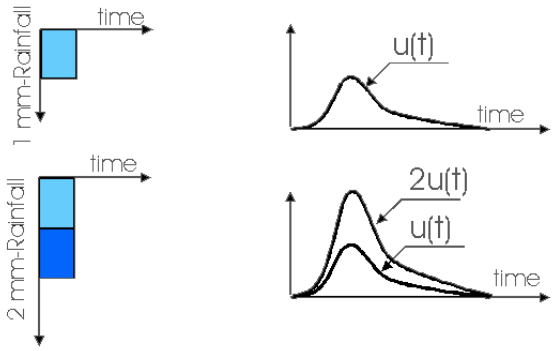

Figure A.2: Linear assumption of unit hydrograph (VICAIRE n.d.)

The linearity property (assumption 5) forms the core of the unit hydrograph theory and its application, with its main characteristics including proportionality (Figure A.2) and additivity, also called superposition (Figure A.3). The proportionality principle can be explained as the unit hydrograph's ordinates are directly proportional to the intensity of rainfall. This means that when the rain intensity is doubled within the same duration, the catchment's runoff hydrograph will be double (Figure A.2) (VICAIRE n.d.; Butler et al. 2018). The superposition property states that with n blocks of excess rainfall, each starting at different times, the catchment's hydrograph is calculated by summing all component hydrographs resulting from each block of rain at its corresponding time (Butler et al. 2018; VICAIRE n.d.).

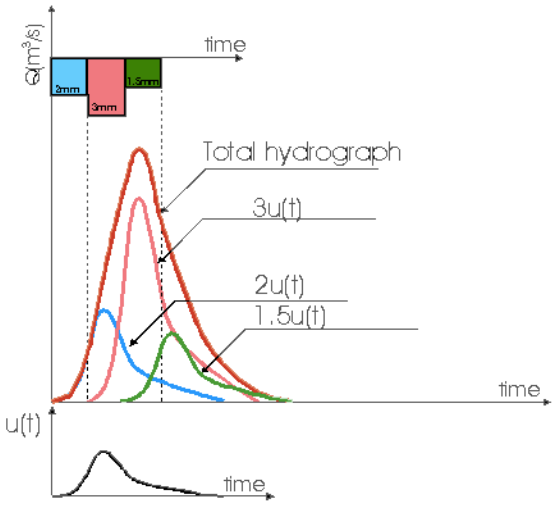

Figure A.3: Additivity principle of unit hydrograph (VICAIRE n.d.)

Regarding its characteristics, there are several parameters that can be used to describe a unit hydrograph (Figure A.4). u_p in the figure describes the peak discharge of the hydrograph, while t_b represents the base time, which is the duration of the unit hydrograph. t_p is the time between the start of the unit hydrograph until it reaches its peak. The concentration time (t_c) is the duration between when the

rain ends and the UH stops. Finally, the lag time (t_{lag}) is the difference in time between the gravity center of the rain and the UH's peak. (VICAIRE n.d.).

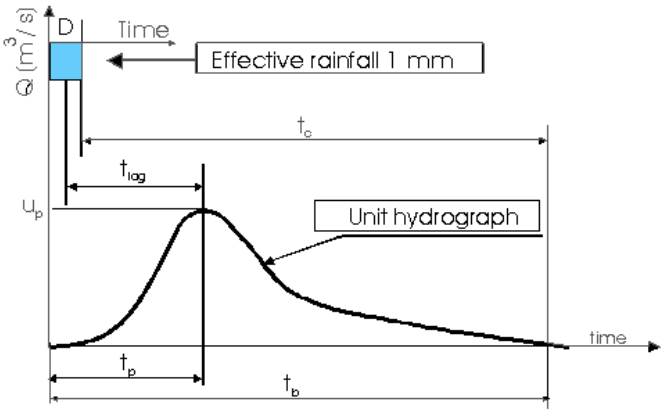

Figure A.4: Characteristics of unit hydrographs (VICAIRE n.d.)

A.2 Instantaneous Unit Hydropgraph

The instantaneous unit hydrograph (IUH) is defined as the unit hydrograph when the effective rainfall has a depth of 1 mm and an infinitesimal duration, which means that the duration of the rain tends towards zero. This type of hydrograph is only theoretical and does not describe a catchment, but instead can be advantageous as it is able to indicate catchment's response to rainfall without having to refer to the rain duration (Ven Chow 1988.). Another advantage of the IUH over the UH is that it does not expect rainfall to be uniformly distributed throughout the catchment. Therefore, it can be said as the visual representation of the catchment's response to rain based on that catchment's physical characteristics such as its shape, length or slope, etc (Mohan and Vijayalakshmi 2008).

This can be re-phrased in terms of system analysis. The system is the catchment, with the rainfall as input and the discharge as output. Then, having an instantaneous input with unit area will result in obtaining the system's impulse response (the transfer function), which in this case is the IUH.

The properties of linearity and time invariance still hold here, allowing the calculation of the discharge using the convolution operation:

$$Q(t) = I(t) * u(t) = \int_0^t I(\tau)u(t-\tau)d\tau \tag{A.1}$$

where $I(t)$ is the rainfall, $u(t)$ is the instantaneous hydrograph, $*$ indicates the convolution operation and $Q(t)$ is the discharge of the catchment.

A.2.1 Considerations Regarding the Discrete Time Case

When performing computer analysis and also in this research, the operations are carried out in a discrete form. This requires reformulating the IUH in discrete time. One method is to simply evaluate the continuous time IUH at each time step. However, doing so would change the total volume discharged by the system. This can be observed by comparing the volume of the continuous time IUH and the discrete time IUH:

$$\int_0^\infty Q(t)dt \neq T\sum_{m=0}^{\infty} Q(mT)$$

where T is the length of each time step and m indicates the current discrete time step, i.e. sampling the continuous time discharge every T seconds.

The second method consists of finding the volume of water that the system discharges during each time step when an instantaneous input of unit volume is given to the system. This will then allow to compute the discrete time convolution between any inflow and the IUH, giving the predicted discharge. The unit of volume chosen is m^3 and the time step is minutes.

Given the Kronecker delta:

$$\delta_k[t] = \begin{cases} 0 & \text{if } t \neq 0 \\ 1 & \text{if } t = 0 \end{cases} \tag{A.2}$$

We have:

$$u[t] = u_0 \cdot \delta_k[t] + u_1 \cdot \delta_k[t-1] + \cdots + u_{-1} \cdot \delta_k[t+1] + u_{-2} \cdot \delta_k[t+2] + \ldots \tag{A.3}$$

$$u_i = \int_{iT}^{(i+1)T} u(t)dt, \quad i = 0, 1, \ldots, -1, -2, \ldots \tag{A.4}$$

$$Q[t] = T\sum_{l=-\infty}^{+\infty} I[l] \cdot H[t-l] \tag{A.5}$$

where T is the duration of a single time step, I is the inflow to the system, Q is the predicted discharge. Equation A.3 explicitly describes how the discrete instantaneous unit hydrograph is composed. Equation A.4 describes how each coefficient is obtained by finding the area under the curve of the continuous time instantaneous unit hydrograph between the beginning of the i-th time step and the beginning of the $(i+1)$-th time step. This is then used in Equation A.5 to find the outflow given any inflow by using a discrete time convolution.

A.2.2 Characteristics of IUH

The IUH should be in the form of a single-peaked runoff hydrograph. Nevertheless, it is also possible that its ordinate's values are negative or in a wavy form (Ven Chow 1988). Additionally, the unit for IUH's ordinate is in T^{-1}, with T is time. Furthermore, the IUH also bears the following characteristics:

$$0 \leq u(t) \leq u(t = t_p) \text{ for } t > 0 \text{ and } t \neq t_p \tag{A.6}$$

$$u(t) = 0 \text{ for } t \leq 0 \tag{A.7}$$

$$u(t) \to 0 \text{ as } t \to \infty \tag{A.8}$$

$$\int_0^\infty u(t)dt = 1 \tag{A.9}$$

B Test to Obtain Storage Constant Using Random Inflow

To test whether the type of different inflow would affect the result obtained for storage constant c, a random inflow time series is set up, also without dry period in between. The random inflow is presented in Figure B.1.

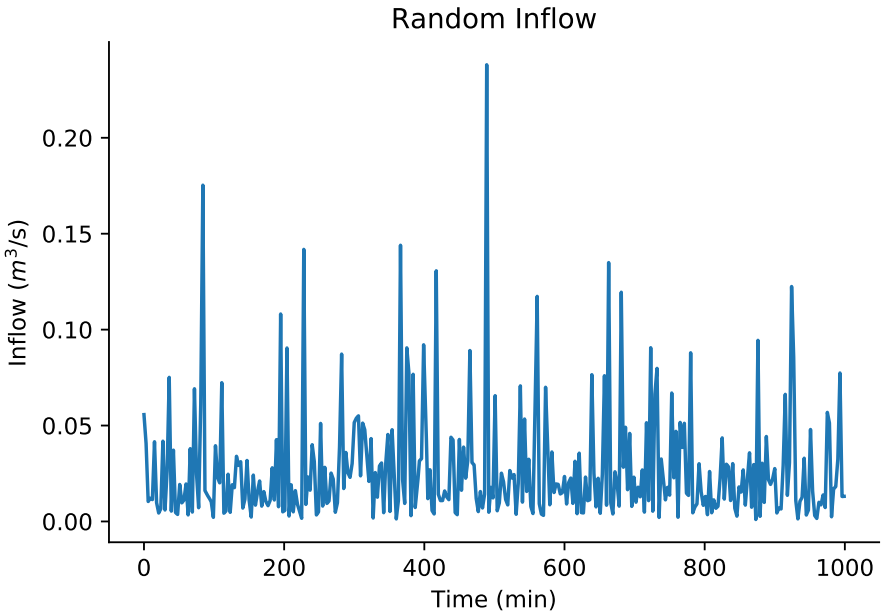

Figure B.1: Random inflow time series

Two examples are examined and compared between using random and dry period excluded inflow. The examples include finding storage constant c for the transport stretch of length $50m$ and slope $1‰$, and length $500m$ and slope $1‰$since this slope often gives the most erroneous results. The results of this comparison is seen in Figure B.2.

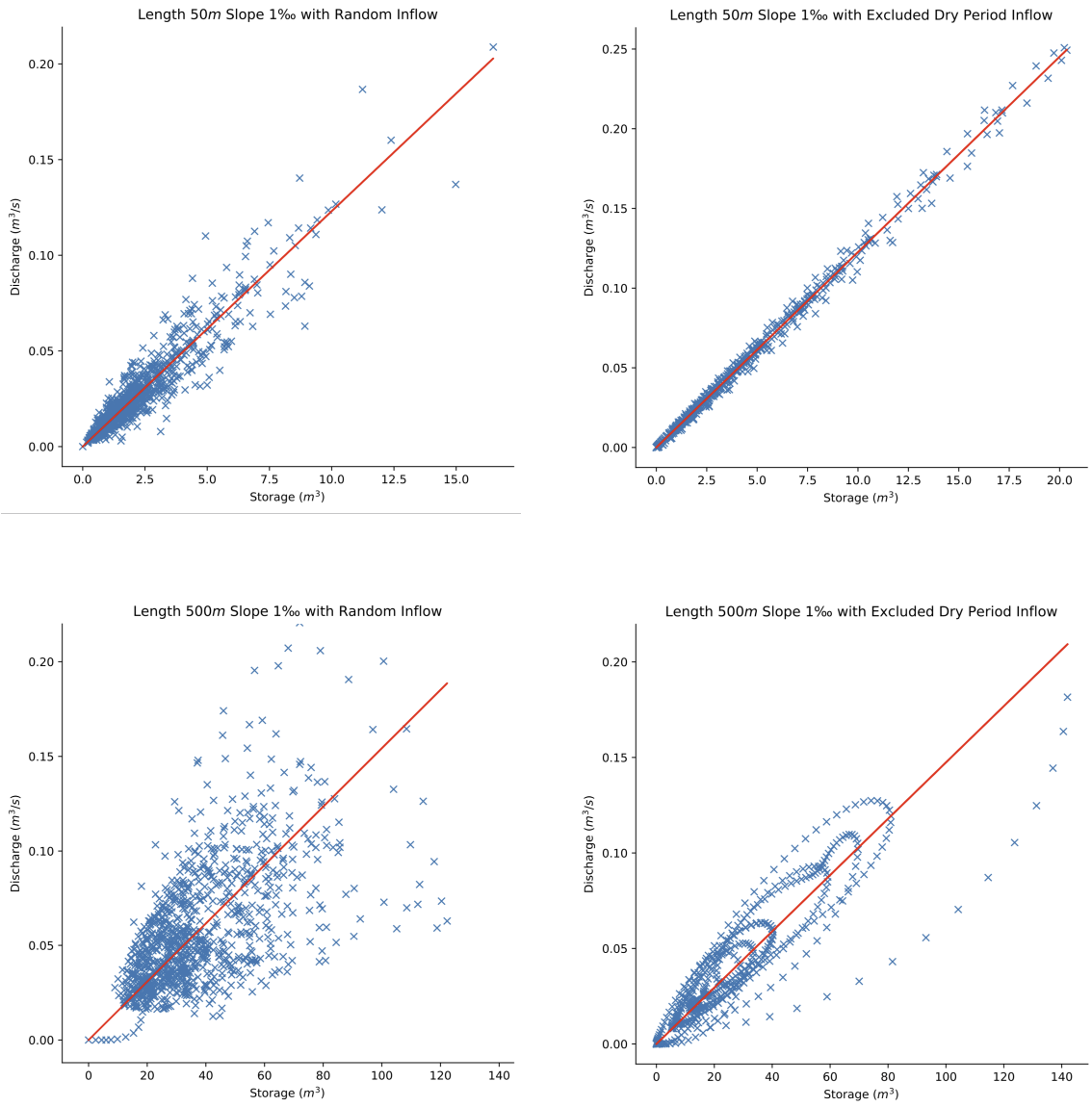

Figure B.2: SQ slopes of transport stretch length $50m$ (up) and $500m$ (down) with slope $1‰$, between using a random inflow (right) and the inflow with dry period exclusion (left)

The resulting storage constant c derived from the two inflow time series is present in Table B.1.

Table B.1: Storage constant c (minute) with random inflow and dry period excluded inflow

	Random inflow	**Dry period excluded inflow**
Length $50m$ slope $1‰$	1.354	1.359
Length $500m$ slope $1‰$	10.804	11.309

As can be seen, the slopes of the SQ plots generated using different inflow series did

not change significantly even though the storage-discharge values for the random inflow spreads more. The derived storage constant c also does not vary much, with a difference of only 30 seconds for the pipe of 500m, which is rather marginal. Therefore, it can be concluded that the storage constant c can be computed using any rain time series.

C Rational Method Dimensioning

C.1 General Cities Parameters

The parameters for the two cities are similar, where each block of house has the size of 360 m^2 (12 m x 30 m). There are 2 inhabitants per house. The small roads have a width of 7 m while that for big roads is 15 m. For Star city, big roads are the one in the middle running from manhole A3 to G5 and one at the bottom from G1 to G9. The big road only runs from L1 to L12 which is at the bottom of the city in the case of Strip city. The runoff coefficient for houses is 0.4 and that of the roads is 0.8 according to typical values of runoff for residential areas (0.3 - 0.7) and for asphalt and concrete paving (0.7 - 0.95), respectively, recommended by Butler et al. 2018. The designed wastewater flow rate for pipe design is 0.003 liters/person/day.

C.2 Slopes and Manholes Design

Manholes are positioned so that they are around 90 m from each other, as recommended from Butler et al. 2018. Horizontally, manhole distances vary from 72 m to 111 m depending on the pipe layouts as seen from ... . All manholes are 67 m of distance on vertical roads at the junction with other pipelines. Catchments are designed so that they have relatively similar areas, consisting of 14 - 16 houses with roads in between, except for the ones in the bottom of the two cities, where it is not possible to do so. Hence, at these positions, the areas of these catchments are half of the normal areas above them.

The pipe slopes are designed as described for Danish standards to satisfy the wastewater flow of 0.003 l/p/s. This number is derived from the Equation C.1, with the values of $f_{day,min}$ of 0.8, $f_{hour,max}$ of 2.0 and $q_{average}$ of 160 l/p/d (Morten Borup 2019).

$$q_{design} = f_{day,min} * f_{hour,max} * q_{average} \tag{C.1}$$

Where q_{design} (liters/person/day) is the designed flow rate of the sewer system, $f_{day,min}$ is the coefficient describing the minimum flow of the day, $f_{hour,max}$ is the coefficient for the maximum flow of the hour and $q_{average}$ is the average flow of wastewater consumed by one person in a day (liters/person/day). For a Danish sewer system, the value of $f_{day,min}$ typically ranges from 0.5 to 0.9, and that of $f_{hour,max}$ is between 1.5 and 2.5 depending on the type of area.

The minimum slope for pipes with q_{design} < 1 l/s is chosen to be 10‰so that self cleansing of pipes can be achieved. The available slope is calculated as the terrain

slope to minimize excavation as much as possible. The difference between manhole's invert and ground levels is kept between 1 m and 5 m for both cities to avoid easy overflowing and deep excavation. All pipes flow into the outlet which delivers the water out of the system.

C.3 Design Storm and Pipe Diameter Dimensionsing

According to Danish design criteria, for combined systems, the drainage network should be full running for 2-year rain without overflowing as suggested by Spildevandskomiteen 2005. The CDS is designed to last 4 hours. The intensity of the storm can be seen in Figure C.1.

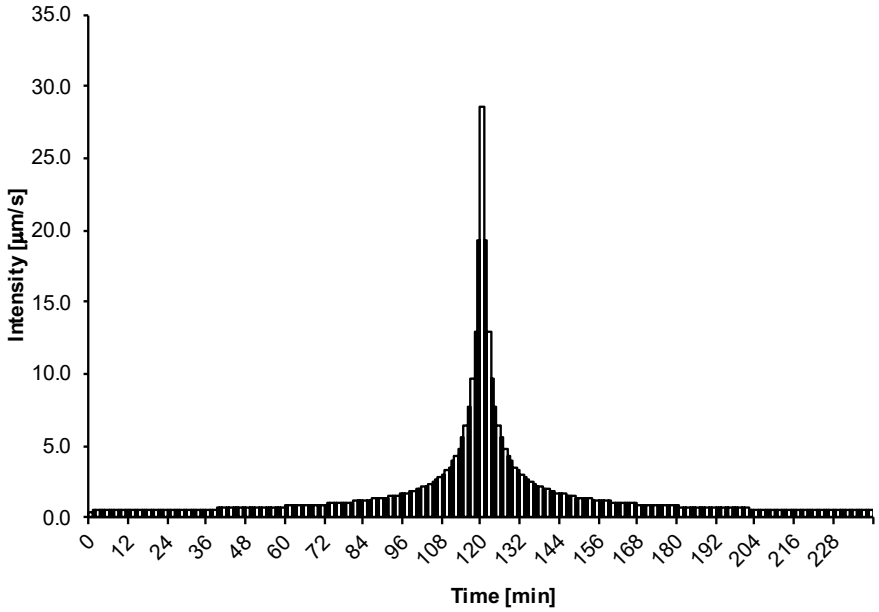

Figure C.1: Rain time series

A regression of the IDF curve is present in Figure C.2, based on which the rain intensity is read to calculate the maximum stormwater flow in a pipe.

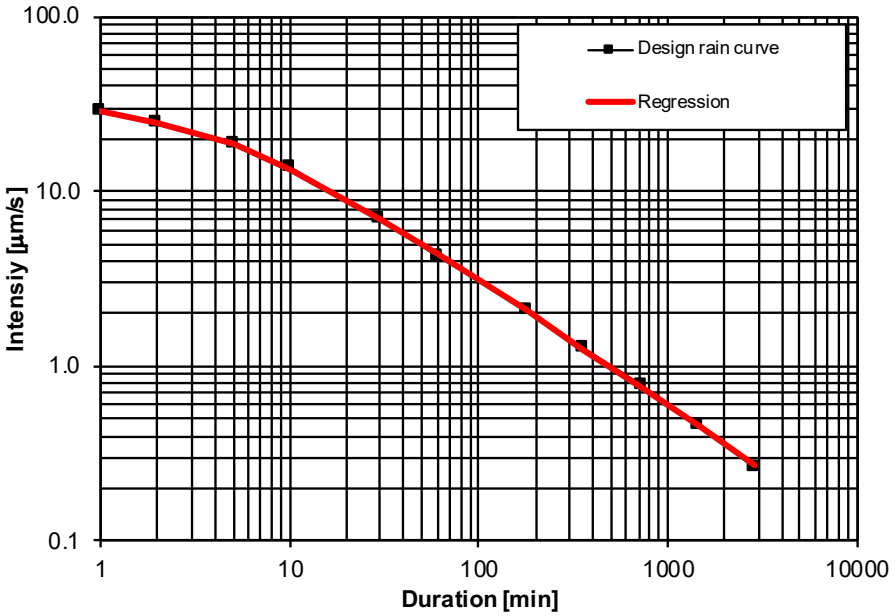

Figure C.2: Rain intensity

Time of entry is selected as 300 seconds (5 minutes), which is in the range of 4 - 7 minutes for storm with return period of 2 years summarized from Butler et al. 2018.

The flow velocity and pipe diameter are calculated based on the Colebrook-White equation. For the pipe diameters, the range of values are 225, 300, 400, 500, 600, $700mm$. Each calculated diameter has to be rounded up to these values since in reality, these are the standard pipe diameters.

C.4 Rational Method Virtual City Design

The detailed dimensioning of each city's parameter based on the Rational method can be found in the following tables.

C.4.1 Star City Design

Table C.1: Rational method table for Star City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
nr.	-	m^2	m^2	m^2	ha	%	m^2	pers	pers	l/s	l/s	m	um/s	m^3/s	m^3/s	‰	mm	m^3/s	m/s	second	second	m	m	m
A1																							27.93	30
	A1-A2	4320	1393	2842.4	0.5713	49.75	2842.4	24	24	0.072	0.072	96	16.59	0.0472	0.0472	10	225	0.052	1.307	73.426	373.426	0.96		
A2																							26.97	29.4
	A2-A3	5040	588	2486.4	0.5628	44.18	5328.8	28	52	0.084	0.156	96	15.56	0.0829	0.0831	10	300	0.111	1.572	61.058	434.484	0.96		
A3																							26.01	28.8
	A3-A4	5040	1593	3290.4	0.6633	49.61	8619.2	28	80	0.084	0.24	111	14.54	0.1253	0.1255	10	400	0.237	1.887	58.809	493.293	1.11		
A4																							24.90	28.11
	A4-A5	5040	588	2486.4	0.5628	44.18	11106	28	108	0.084	0.324	96	13.51	0.1500	0.1503	10	400	0.237	1.887	50.862	544.155	0.96		
A5																							23.94	27.51
H1																							24.61	28.04
	H1-A5	1800	210	888	0.2010	44.18	888	10	10	0.03	0.03	67	16.59	0.0147	0.0148	10	225	0.052	1.307	51.245	351.245	0.67		
A5																							23.94	27.51
	A5-B5	3600	1309	2487.2	0.4909	50.67	14481	20	138	0.06	0.414	67	13.18	0.1909	0.1913	10	400	0.237	1.887	35.497	579.653	0.67		
B5																							23.27	26.99
B1																							27.26	29.48
	B1-B2	4320	1393	2842.4	0.5713	49.75	2842.4	24	24	0.072	0.072	96	16.59	0.0472	0.0472	10	225	0.052	1.307	73.426	373.426	0.96		
B2																							26.30	28.88
	B2-B3	5040	588	2486.4	0.5628	44.18	5328.8	28	52	0.084	0.156	96	15.56	0.0829	0.0831	10	300	0.111	1.572	61.058	434.484	0.96		
B3																							25.34	28.28
	B3-B4	5040	1593	3290.4	0.6633	49.61	8619.2	28	80	0.084	0.24	111	14.54	0.1253	0.1255	10	400	0.237	1.887	58.809	493.293	1.11		
B4																							24.23	27.59
	B4-B5	5040	588	2486.4	0.5628	44.18	11106	28	108	0.084	0.324	96	13.51	0.1500	0.1503	10	400	0.237	1.887	50.862	544.155	0.96		
B5																							23.27	26.99
	B5-C5	3600	1309	2487.2	0.4909	50.67	28074	20	266	0.06	0.798	67	12.86	0.3611	0.3619	10	500	0.427	2.172	30.841	610.493	0.67		
C5																							22.60	26.46
C1																							26.59	28.95
	C1-C2	4320	1393	2842.4	0.5713	49.75	2842.4	24	24	0.072	0.072	96	16.59	0.0472	0.0472	10	225	0.052	1.307	73.426	373.426	0.96		
C2																							25.63	28.35
	C2-C3	5040	588	2486.4	0.5628	44.18	5328.8	28	52	0.084	0.156	96	15.56	0.0829	0.0831	10	300	0.111	1.572	61.058	434.484	0.96		
C3																							24.67	27.76
	C3-C4	5040	1593	3290.4	0.6633	49.61	8619.2	28	80	0.084	0.24	111	14.54	0.1253	0.1255	10	400	0.237	1.887	58.809	493.293	1.11		
C4																							23.56	27.06
	C4-C5	5040	588	2486.4	0.5628	44.18	11106	28	108	0.084	0.324	96	13.51	0.1500	0.1503	10	400	0.237	1.887	50.862	544.155	0.96		
C5																							22.60	26.46
	C5-D6	3600	1309	2487.2	0.4909	50.67	41666	20	394	0.06	1.182	67	12.86	0.5359	0.5371	9.198	600	0.660	2.335	28.689	639.182	0.616		

Table C.1: Rational method table for Star City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
D6																							21.98	25.94
D1																							26.36	28.43
	D1-D2	4320	1393	2842.4	0.5713	49.75	2842.4	24	24	0.072	0.072	96	16.59	0.0472	0.0472	10	225	0.052	1.307	73.426	373.426	0.96		
D2																							25.40	27.83
	D2-D3	5040	588	2486.4	0.5628	44.18	5328.8	28	52	0.084	0.156	96	15.56	0.0829	0.0831	10	300	0.111	1.572	61.058	434.484	0.96		
D3																							24.44	27.23
	D3-E6	0	0	0	0		5328.8	0	52	0	0.156	67	14.54	0.0775	0.0776	10	300	0.111	1.572	42.614	477.098	0.67		
E6																							23.77	26.71
D4																							23.66	26.99
	D4-D5	5040	1593	3290.4	0.6633	49.61	3290.4	28	28	0.084	0.084	72	16.59	0.0546	0.0547	10	300	0.111	1.572	45.794	345.794	0.72		
D5																							22.94	26.54
	D5-D6	5040	588	2486.4	0.5628	44.18	5776.8	28	56	0.084	0.168	96	15.56	0.0899	0.0901	10	300	0.111	1.572	61.058	406.852	0.96		
D6																							21.98	25.94
	D6-E9	3600	1309	2487.2	0.4909	50.67	49930	20	470	0.06	1.41	67	12.54	0.6261	0.6275	8.422	600	0.632	2.234	29.996	669.178	0.564		
E9																							21.42	25.42
E1																							27.00	29.1
	E1-E2	1800	420	1056	0.2220	47.57	1056	10	10	0.03	0.03	96	16.59	0.0175	0.0176	10	225	0.052	1.307	73.426	373.426	0.96		
E2																							26.65	28.5
	E2-E3	2160	504	1267.2	0.2664	47.57	2323.2	12	22	0.036	0.066	96	15.56	0.0362	0.0362	10	225	0.052	1.307	73.426	446.852	0.96		
E3																							25.69	27.91
	E3-F3	0	0	0	0		2323.2	0	22	0	0.066	67	14.54	0.0338	0.0338	10	225	0.052	1.307	51.245	498.097	0.67		
F3																							25.02	27.38
E4																							25.45	27.76
	E4-E5	6120	1393	3562.4	0.7513	47.42	3562.4	34	34	0.102	0.102	72	16.59	0.0591	0.0592	10	300	0.111	1.572	45.794	345.794	0.72		
E5																							24.73	27.31
	E5-E6	5040	588	2486.4	0.5628	44.18	6048.8	28	62	0.084	0.186	96	15.56	0.0941	0.0943	10	300	0.111	1.572	61.058	406.852	0.96		
E6																							23.77	26.71
	E6-F6	0	0	0	0		11378	0	114	0	0.342	67	13.51	0.1537	0.1540	10	400	0.237	1.887	35.497	512.595	0.67		
F6																							23.10	26.18
E7																							23.10	26.46
	E7-E8	5040	1593	3290.4	0.6633	49.61	3290.4	28	28	0.084	0.084	72	16.59	0.0546	0.0547	10	300	0.111	1.572	45.794	345.794	0.72		
E8																							22.38	26.02
	E8-E9	5040	588	2486.4	0.5628	44.18	5776.8	28	56	0.084	0.168	96	15.56	0.0899	0.0901	10	300	0.111	1.572	61.058	406.852	0.96		
E9																							21.42	25.42
	E9-F9	5400	1309	3207.2	0.6709	47.80	58914	30	556	0.09	1.668	67	12.22	0.7197	0.7214	7.910	700	0.917	2.380	28.151	697.329	0.53		
F9																							20.89	24.89

Table C.1: Rational method table for Star City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
E10																							21.43	24.77
	E10-E11	3960	924	2323.2	0.4884	47.57	2323.2	22	22	0.066	0.066	96	16.59	0.0386	0.0386	10	225	0.052	1.307	73.426	373.426	0.96		
E11																							20.47	24.18
	E11-F11	0	0	0	0		2323.2	0	22	0	0.066	67	15.56	0.0362	0.0362	10	225	0.052	1.307	51.245	424.671	0.67		
F11																							19.80	23.65
F1																							26.70	28.58
	F1-F2	3600	420	1776	0.402	44.18	1776	20	20	0.06	0.06	96	16.59	0.0295	0.0295	10	225	0.052	1.307	73.426	373.426	0.96		
F2																							25.98	27.98
	F2-F3	4320	504	2131.2	0.4824	44.18	3907.2	24	44	0.072	0.132	96	15.56	0.0608	0.0609	10	300	0.111	1.572	61.058	434.484	0.96		
F3																							25.02	27.38
	F3-G3	0	0	0	0		6230.4	0	66	0	0.198	67	13.51	0.0841	0.0843	10	300	0.111	1.572	42.614	540.711	0.67		
G3																							24.35	26.86
F4																							24.78	27.23
	F4-F5	7920	1393	4282.4	0.9313	45.98	4282.4	44	44	0.132	0.132	72	16.59	0.0711	0.0712	10	300	0.111	1.572	45.794	345.794	0.72		
F5																							24.06	26.78
	F5-F6	5040	588	2486.4	0.5628	44.18	6768.8	28	72	0.084	0.216	96	15.56	0.1054	0.1056	10	300	0.111	1.572	61.058	406.852	0.96		
F6																							23.10	26.18
	F6-G5	0	0	0	0		18146	0	186	0	0.558	67	13.51	0.2451	0.2456	10	500	0.427	2.172	30.841	543.436	0.67		
G5																							22.43	25.66
F7																							22.57	25.94
	F7-F8	5040	1593	3290.4	0.6633	49.61	3290.4	28	28	0.084	0.084	72	16.59	0.0546	0.0547	10	300	0.111	1.572	45.794	345.794	0.72		
F8																							21.85	25.5
	F8-F9	5040	588	2486.4	0.5628	44.18	5776.8	28	56	0.084	0.168	96	15.56	0.0899	0.0901	10	300	0.111	1.572	61.058	406.852	0.96		
F9																							20.89	24.89
	F9-G7	7200	1309	3927.2	0.8509	46.15	68618	40	652	0.12	1.956	67	12.22	0.8383	0.8403	7.761	700	0.908	2.360	28.390	725.719	0.52		
G7																							20.37	24.37
F10																							20.76	24.25
	F10-F11	7920	924	3907.2	0.8844	44.18	3907.2	44	44	0.132	0.132	96	16.59	0.0648	0.0650	10	300	0.111	1.572	61.058	361.058	0.96		
F11																							19.80	23.65
	F11-G9	0	0	0	0		6230.4	0	66	0	0.198	67	14.54	0.0906	0.0908	10	300	0.111	1.572	42.614	467.285	0.67		
G9																							19.13	23.13
G1																							26.27	28
	G1-G2	1800	900	1440	0.27	53.33	1440	10	10	0.03	0.03	96	16.59	0.0239	0.0239	10	225	0.052	1.307	73.426	373.426	0.96		
G2																							25.31	27.46
	G2-G3	2160	1080	1728	0.324	53.33	3168	12	22	0.036	0.066	96	15.56	0.0493	0.0494	10	225	0.052	1.307	73.426	446.852	0.96		
G3																							24.35	26.86
	G3-G4	3960	2295	3420	0.6255	54.68	12818	22	110	0.066	0.33	96	13.18	0.1690	0.1693	10	400	0.237	1.887	50.862	591.573	0.96		

Table C.1: Rational method table for Star City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
G4																							23.39	26.26
	G4-G5	2520	1260	2016	0.378	53.33	14834	14	124	0.042	0.372	96	12.54	0.1860	0.1864	10	400	0.237	1.887	50.862	642.435	0.96		
G5																							22.43	25.66
	G5-G6	2520	1935	2556	0.4455	57.37	35537	14	324	0.042	0.972	111	12.22	0.4341	0.4351	10	600	0.689	2.435	45.578	688.013	1.11		
G6																							21.32	24.97
	G6-G7	2520	1260	2016	0.378	53.33	37553	14	338	0.042	1.014	96	12.22	0.4588	0.4598	9.931	600	0.686	2.427	39.558	727.571	0.953		
G7																							20.37	24.37
	G7-G8	3600	2115	3132	0.5715	54.80	109303	20	1010	0.06	3.03	103	11.89	1.3001	1.3031	6.214	700	0.812	2.110	48.812	774.531	0.64		
G8																							19.73	23.73
	G8-G9	3960	1980	3168	0.594	53.33	112471	22	1032	0.066	3.096	96	11.57	1.3016	1.3047	6.25	700	0.815	2.117	45.348	819.879	0.6		
G9																							19.13	23.13
	G9-outlet	0	0	0	0		118702	0	1032	0	3.096	96	11.25	1.3354	1.3385	6.25	700	0.815	2.117	45.348	865.227	0.6		
Outlet																							18.53	22.53

C.4.2 Strip City Design

Table C.2: Rational method computations for Strip City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
nr.	-	m^2	m^2	m^2	ha	%	m^2	pers	pers	l/s	l/s	m	um/s	m^3/s	m^3/s	‰	mm	m^3/s	m/s	second	second	m	m	m
I1																							28.28	30
	I1-I2	5040	588	2486	0.563	44.18	2486.4	28	28	0.084	0.084	96	16.5946	0.0413	0.0413	10	225	0.052	1.307	73.426	373.426	0.96		
I2																							27.50	29.40
	I2-I3	5040	588	2486	0.563	44.18	4972.8	28	56	0.084	0.168	96	15.5649	0.0774	0.0776	10	300	0.111	1.572	61.058	434.484	0.96		
I3																							26.79	28.80
	I3-K3	0	0	0	0	0	4972.8	0	56	0	0.168	67	14.5352	0.0723	0.0724	10	300	0.111	1.572	42.614	477.098	0.67		
K3																							26.12	28.28
I4																							26.48	28.61
	I4-I5	5040	847	2694	0.589	45.76	2693.6	28	28	0.084	0.084	72	16.5946	0.0447	0.0448	10	225	0.052	1.307	55.069	355.069	0.72		
I5																							25.76	28.16
	I5-I6	5040	588	2486	0.563	44.18	5180	28	56	0.084	0.168	96	15.5649	0.0806	0.0808	10	300	0.111	1.572	61.058	416.128	0.96		
I6																							24.80	27.56
	I6-K6	0	0	0	0	0	5180	0	56	0	0.168	67	14.5352	0.0753	0.0755	10	300	0.111	1.572	42.614	458.741	0.67		
K6																							24.13	27.04
I7																							24.49	27.37
	I7-I8	5760	931	3049	0.669	45.57	3048.8	32	32	0.096	0.096	72	16.5946	0.0506	0.0507	10	225	0.052	1.307	55.069	355.069	0.72		
I8																							23.77	26.92
	I8-I9	5760	672	2842	0.643	44.18	5890.4	32	64	0.096	0.192	96	15.5649	0.0917	0.0919	10	300	0.111	1.572	61.058	416.128	0.96		
I9																							22.81	26.32
	I9-K9	0	0	0	0	0	5890.4	0	64	0	0.192	67	14.5352	0.0856	0.0858	10	300	0.111	1.572	42.614	458.741	0.67		
K9																							22.14	25.80
I10																							22.81	26.13
	I10-I11	5760	931	3049	0.669	45.57	3048.8	32	32	0.096	0.096	72	16.5946	0.0506	0.0507	10	225	0.052	1.307	55.069	355.069	0.72		
I11																							22.09	25.68
	I11-I12	5760	672	2842	0.643	44.18	5890.4	32	64	0.096	0.192	96	15.5649	0.0917	0.0919	10	300	0.111	1.572	61.058	416.128	0.96		
I12																							21.13	25.08
	I12-K12	0	0	0	0	0	5890.4	0	64	0	0.192	67	14.5352	0.0856	0.0858	10	300	0.111	1.572	42.614	458.741	0.67		
K12																							20.46	24.56
I13																							21.35	24.89
	I13-I14	5760	931	3049	0.669	45.57	3048.8	32	32	0.096	0.096	72	16.5946	0.0506	0.0507	10	225	0.052	1.307	55.069	355.069	0.72		
I14																							20.63	24.44
	I14-I15	5040	588	2486	0.563	44.18	5535.2	28	60	0.084	0.18	96	15.5649	0.0862	0.0863	10	300	0.111	1.572	61.058	416.128	0.96		
I15																							19.67	23.84
	I15-K15	0	0	0	0	0	5535.2	0	60	0	0.18	67	14.5352	0.0805	0.0806	10	300	0.111	1.572	42.614	458.741	0.67		
K15																							19.00	23.31

Table C.2: Rational method computations for Strip City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
I16																							20.07	23.64
	I16-I17	5760	931	3049	0.669	45.57	3048.8	32	32	0.096	0.096	72	16.5946	0.0506	0.0507	10	225	0.052	1.307	55.069	355.069	0.72		
I17																							19.35	23.20
	I17-I18	9360	1092	4618	1.045	44.18	7666.4	52	84	0.156	0.252	96	15.5649	0.1193	0.1196	10	400	0.237	1.887	50.862	405.931	0.96		
I18																							18.39	22.60
	I18-K18	0	0	0	0	0	7666.4	0	84	0	0.252	67	14.5352	0.1114	0.1117	10	400	0.237	1.887	35.497	441.429	0.67		
K18																							17.72	22.07
K1																							27.70	29.48
	K1-K2	5040	588	2486	0.563	44.18	2486.4	28	28	0.084	0.084	96	16.5946	0.0413	0.0413	10	225	0.052	1.307	73.426	373.426	0.96		
K2																							27.08	28.88
	K2-K3	5040	588	2486	0.563	44.18	4972.8	28	56	0.084	0.168	96	15.5649	0.0774	0.0776	10	300	0.111	1.572	61.058	434.484	0.96		
K3																							26.12	28.28
	K3-L3	0	0	0	0	0	9945.6	0	112	0	0.336	67	13.5054	0.1343	0.1347	10	400	0.237	1.887	35.497	512.595	0.67		
L3																							25.45	27.76
K4																							25.81	28.09
	K4-K5	5040	1057	2862	0.61	46.93	2861.6	28	28	0.084	0.084	72	16.5946	0.0475	0.0476	10	225	0.052	1.307	55.069	355.069	0.72		
K5																							25.09	27.64
	K5-K6	5040	588	2486	0.563	44.18	5348	28	56	0.084	0.168	96	15.5649	0.0832	0.0834	10	300	0.111	1.572	61.058	416.128	0.96		
K6																							24.13	27.04
	K6-L5	0	0	0	0	0	10528	0	112	0	0.336	67	13.5054	0.1422	0.1425	10	400	0.237	1.887	35.497	494.239	0.67		
L5																							23.46	26.51
K7																							23.82	26.84
	K7-K8	5760	1141	3217	0.69	46.61	3216.8	32	32	0.096	0.096	72	16.5946	0.0534	0.0535	10	300	0.111	1.572	45.794	345.794	0.72		
K8																							23.10	26.40
	K8-K9	5760	672	2842	0.643	44.18	6058.4	32	64	0.096	0.192	96	15.5649	0.0943	0.0945	10	300	0.111	1.572	61.058	406.852	0.96		
K9																							22.14	25.80
	K9-L7	0	0	0	0	0	11948.8	0	128	0	0.384	67	14.5352	0.1737	0.1741	10	400	0.237	1.887	35.497	494.239	0.67		
L7																							21.47	25.27
K10																							22.14	25.60
	K10-K11	5760	1141	3217	0.69	46.61	3216.8	32	32	0.096	0.096	72	16.5946	0.0534	0.0535	10	300	0.111	1.572	45.794	345.794	0.72		
K11																							21.42	25.15
	K11-K12	5760	672	2842	0.643	44.18	6058.4	32	64	0.096	0.192	96	15.5649	0.0943	0.0945	10	300	0.111	1.572	61.058	406.852	0.96		
K12																							20.46	24.56
	K12-L9	0	0	0	0	0	11948.8	0	128	0	0.384	67	14.5352	0.1737	0.1741	10	400	0.237	1.887	35.497	494.239	0.67		
L9																							19.79	24.03
K13																							20.68	24.36
	K13-K14	5760	1141	3217	0.69	46.61	3216.8	32	32	0.096	0.096	72	16.5946	0.0534	0.0535	10	300	0.111	1.572	45.794	345.794	0.72		

Table C.2: Rational method computations for Strip City

Manhole	Pipe	Housing area, A	Road area	Total Reduced area Ar	Total area	% impervious	Sum of reduced areas	Inhabitants	Sum of inhabitants	Wastewater flow Qs	Sum of wastewater Qs	Pipe length, L	Rain intensity	Max stormwater flow	Max total (SW+WW) flow	Chosen slope, Ib	Dimension, d	Flow, full pipe, Qf	Velocity full pipe, vf	Flow time, tf	Time of concentration	Level difference	Invert level	Ground level
K14																							19.96	23.91
	K14-K15	5040	588	2486	0.563	44.18	5703.2	28	60	0.084	0.18	96	15.5649	0.0888	0.0889	10	300	0.111	1.572	61.058	406.852	0.96		
K15																							19.00	23.31
	K15-L11	0	0	0	0	0	11238.4	0	120	0	0.36	67	13.5054	0.1518	0.1521	10	400	0.237	1.887	35.497	494.239	0.67		
L11																							18.33	22.79
K16																							19.40	23.12
	K16-K17	5760	1141	3217	0.69	46.61	3216.8	32	32	0.096	0.096	72	16.5946	0.0534	0.0535	10	225	0.052	1.307	55.069	355.069	0.72		
K17																							18.68	22.67
	K17-K18	9360	2184	5491	1.154	47.57	8708	52	84	0.156	0.252	96	15.5649	0.1355	0.1358	10	400	0.237	1.887	50.862	405.931	0.96		
K18																							17.72	22.07
	K18-L13	0	0	0	0	0	16374.4	0	168	0	0.504	67	14.5352	0.2380	0.2385	10	500	0.427	2.172	30.841	472.269	0.67		
L13																							17.05	21.55
L1																							27.20	28.95
	L1-L2	2520	1260	2016	0.378	53.33	2016	14	14	0.042	0.042	96	16.5946	0.0335	0.0335	10	225	0.052	1.307	73.426	373.426	0.96		
L2																							26.41	28.35
	L2-L3	2520	1260	2016	0.378	53.33	4032	14	28	0.042	0.084	96	15.5649	0.0628	0.0628	10	300	0.111	1.572	61.058	434.484	0.96		
L3																							25.45	27.76
	L3-L4	2520	1575	2268	0.41	55.38	16245.6	14	154	0.042	0.462	103	13.1833	0.2142	0.2146	10	400	0.237	1.887	54.571	567.166	1.03		
L4																							24.42	27.11
	L4-L5	2520	1260	2016	0.378	53.33	18261.6	14	168	0.042	0.504	96	12.8611	0.2349	0.2354	10	500	0.427	2.172	44.189	611.355	0.96		
L5																							23.46	26.51
	L5-L6	2880	1755	2556	0.464	55.15	31345.6	16	296	0.048	0.888	103	12.5389	0.3930	0.3939	10	500	0.427	2.172	47.412	658.767	1.03		
L6																							22.43	25.87
	L6-L7	2880	1440	2304	0.432	53.33	33649.6	16	312	0.048	0.936	96	12.2168	0.4111	0.4120	10	500	0.427	2.172	44.189	702.956	0.96		
L7																							21.47	25.27
	L7-L8	2880	1755	2556	0.464	55.15	48154.4	16	456	0.048	1.368	103	11.8946	0.5728	0.5741	8.55	600	0.636	2.251	45.759	748.715	0.88		
L8																							20.59	24.63
	L8-L9	2880	1440	2304	0.432	53.33	50458.4	16	472	0.048	1.416	96	11.5725	0.5839	0.5853	8.40	600	0.631	2.231	43.030	791.745	0.81		
L9																							19.79	24.03
	L9-L10	2880	1755	2556	0.464	55.15	64963.2	16	616	0.048	1.848	103	11.2503	0.7309	0.7327	7.36	700	0.885	2.298	44.815	836.560	0.76		
L10																							19.03	23.39
	L10-L11	2520	1260	2016	0.378	53.33	66979.2	14	630	0.042	1.89	96	11.2503	0.7535	0.7554	7.27	700	0.879	2.284	42.028	878.588	0.70		
L11																							18.33	22.79
	L11-L12	2880	1755	2556	0.464	55.15	80773.6	16	766	0.048	2.298	103	10.9281	0.8827	0.8850	6.60	800	1.188	2.364	43.566	922.155	0.68		
L12																							17.65	22.15
	L12-L13	4680	2340	3744	0.702	53.33	100892	26	960	0.078	2.88	96	10.6060	1.0701	1.0729	6.25	800	1.156	2.300	41.733	963.888	0.6		
L13																							17.05	21.55
	L13-outlet	0	0	0	0	0	100892	0	960	0	2.88	96	10.6060	1.0701	1.0729	6.25	800	1.156	2.300	41.733	1005.621	0.6		
Outlet																							16.45	20.95

D Parameters of Virtual Cities Compartments

Table D.1: Number of pipes and area of each compartment in Star city

Scenario	Compartment	Pipes	Area (ha)
Coarse	Coarse	53	24.7197
Medium	Compartment 1	20	7.3581
	Compartment 2	25	14.4453
	Compartment 3	4	1.3728
	Transport stretch	4	1.5435
Fine	Compartment 1	6	3.0521
	Compartment 2	5	2.8511
	Compartment 3	5	2.8511
	Compartment 4	9	2.5923
	Compartment 5	11	4.7658
	Compartment 6	11	6.6405
	Compartment 7	6	1.9668

Table D.2: Number of pipes and area of each compartment in Strip city

Scenario	Compartment	Pipes	Area (ha)
Coarse	Coarse	49	21.2515
Medium	Compartment 1	25	10.1233
	Compartment 2	24	11.1282
Fine	Compartment 1	9	3.4167
	Compartment 2	8	3.1655
	Compartment 3	8	3.5411
	Compartment 4	8	3.5411
	Compartment 5	8	3.3263
	Compartment 6	8	4.2608

E SQ Functions for All Slopes and Lengths

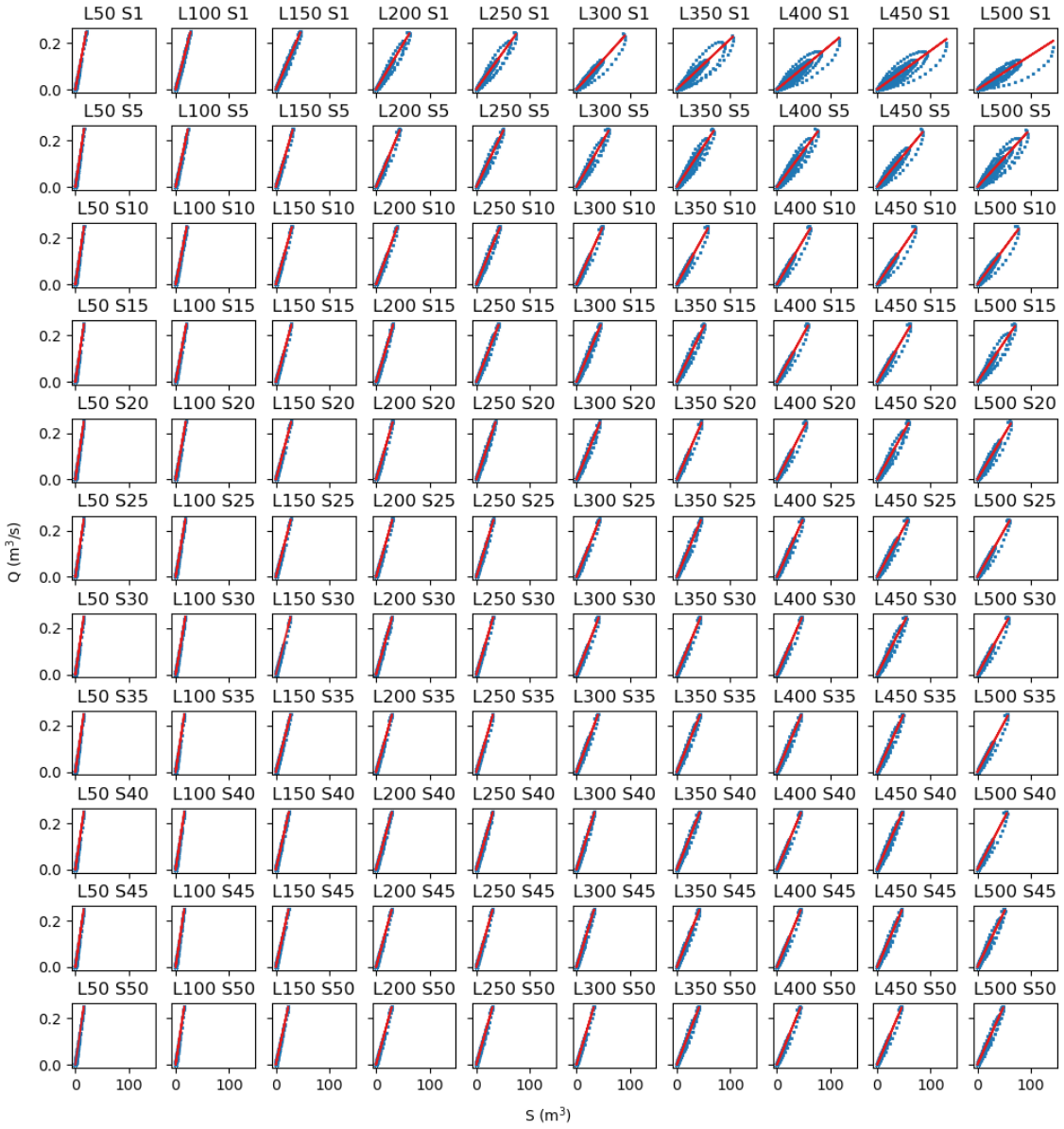

Figure E.1: SQ plots of all slopes and lengths based on k and n ("L" stands for length (m) and "S" stands for slope (‰))

F Virtual Cities Simulation

F.1 PEP, PDIFF and PTDIFF

Table F.1: Mean PEP, $PDIFF$, $PTDIFF$ and their standard errors for Strar City

	PEP mean (%)	**PEP standard error**	**PDIFF mean (m^3/s)**	**PDIFF standard error**	**PTDIFF mean (min)**	**PTDIFF standard error**
Coarse	14.1652	1.7994	0.0153	0.0024	1.1143	0.5133
Medium	4.7984	1.8505	0.0066	0.0017	1.0000	0.4434
Fine	-1.0083	1.8682	0.0016	0.0013	0.1714	0.4230
Full network	-17.2919	2.5226	-0.0135	0.0017	2.3143	0.3752

Table F.2: Mean PEP, $PDIFF$, $PTDIFF$ and their standard errors for Strip City

	PEP mean (%)	**PEP standard error**	**PDIFF mean (m^3/s)**	**PDIFF standard error**	**PTDIFF mean (min)**	**PTDIFF standard error**
Coarse	33.9236	2.4739	0.0262	0.0043	-2.8519	0.5840
Medium	3.3016	2.2916	0.0032	0.0011	2.1852	0.5848
Fine	-0.4271	2.3760	0.0005	0.0009	1.3333	0.6086
Full network	-14.8345	3.4618	-0.0101	0.0017	2.7037	0.5252

G Test with Increased Initial Storage SQ Function

G.1 Method of the Piecewise Linear SQ Model

In this model, the SQ relationship is optimized directly through using the *surrogate model engine for water networks by DTU* (M. Borup 2018) for the transport stretch with different lengths and slopes. The values of lengths and slopes as well as MU+ inflow used as input (Figure 3.2 with dry periods) are the same as the ones described in section 4.1.

The process is that for each length and slope combination of the transport stretch, 2 parameters are optimized in the SQ model. The first parameter is the initial storage S_0, which indicates at which point the model should start discharging more water. S_0 is tested between values of $0m^3$ and $1800m^3$ with a step of $25m^3$. When S_0 is 0, the SQ function becomes linear. The second parameter to be optimized is the slope of the SQ function $(1/c)$ when the storage exceeds S_0. This slope is tested between 0.01 and 1 with a step of 0.01. At storage between 0 and S_0, the value of q_0 is set to be at $0.01m^3/s$, to allow water to still empty out of the system, but only at a very slow rate.

The model tests each combination of S_0 and slope $1/c$ for each transport stretch length and slope and produces a discharge that is compared to the discharge from MU+, using RMSE with the exclusion of dry period (as explained in subsubsection 4.1.2.1). Then, the S_0 and slope $1/c$ combinations that results in the smallest RMSE are chosen to describe that transport stretch based on its length and slope.

G.2 Results of the Piecewise Linear SQ Model

G.2.1 Transport Stretch Optimization with Piecewise Linear SQ Model

G.2.1.1 Values of Initial Storage and Storage Constant

The values of optimized S_0 and c are present in Figure G.1. Even though the optimization is done for the slope $1/c$, the values of c are shown since c makes more sense as a concept of delay, and it is also easier to compare with the storage constant c achieved from the Nash model using a single linear SQ function.

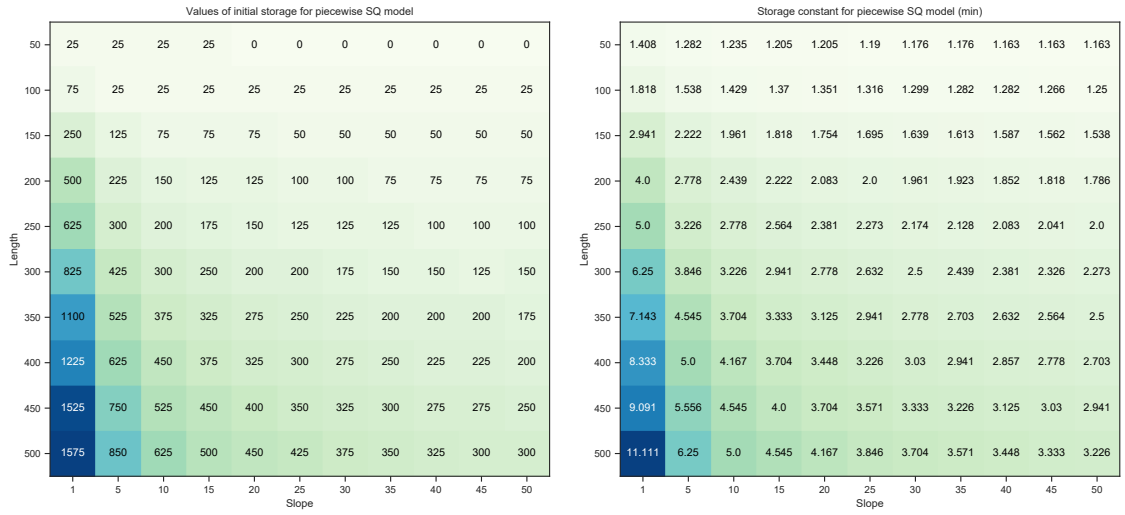

Figure G.1: Values of optimized S_0 (m^3) and storage constant c (minute) for piecewise linear SQ model

As expected, S_0 increases when the pipe becomes longer and when the slope is flatter, meaning that extra delay should be added for these pipes, which agrees with the motivation for why this model is tested (section 5.3). This is understandable as the most delay happens in these situations, hence requires the model to force the delay further and the model needs to "wait" longer before discharging. For the shortest length tested, $50m$, at slopes larger and equal $20‰$, S_0 is 0, meaning that in this case, the best model is the single linear SQ function. The reason is that this transport stretch is short enough with quite high slope, so nearly no delay is required. Once the inflow enters the transport stretch, it will be discharged nearly instantly.

The storage constant c also follows the expected trend, being higher with flatter and longer pipes. This trend is similar to that in Figure 5.6. In fact, the values of c varies very little compared to subsubsection 4.1.2.2, while being identical to Figure 6.1. It means that the storage constant still remains the same, with only some extra initial storage needs to be added at the beginning of the rain to provide proper delay. Other than that, the system still behaves similarly.

G.2.1.2 RMSE of Piecewise Linear Model

Figure G.2 shows the RMSE between the MU+ model discharge and the piecewise SQ model discharge, when dry periods are excluded from calculation.

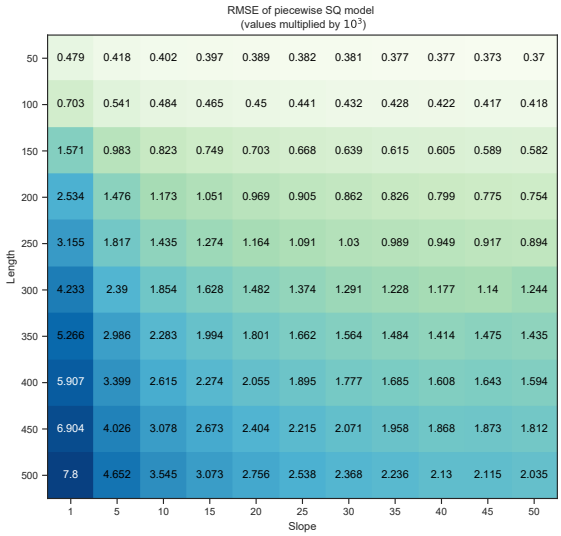

Figure G.2: RMSE between MU+ discharge and piecewise SQ model discharge (values multiplied by 10^3

These errors are smaller than that of the Figure 5.9, proving that it is slightly better at computing discharge when the intial storage is added. Other than that, the trend of RMSE still is similar to that of the single linear SQ model, with longer and flatter pipes experiencing bigger errors.

G.2.1.3 Discharge Simulation

Figure G.3 shows the comparison of discharges between MU+ and conceptual models such as the Nash model, the single linear SQ model and the piecewise SQ model with increased initial storage.

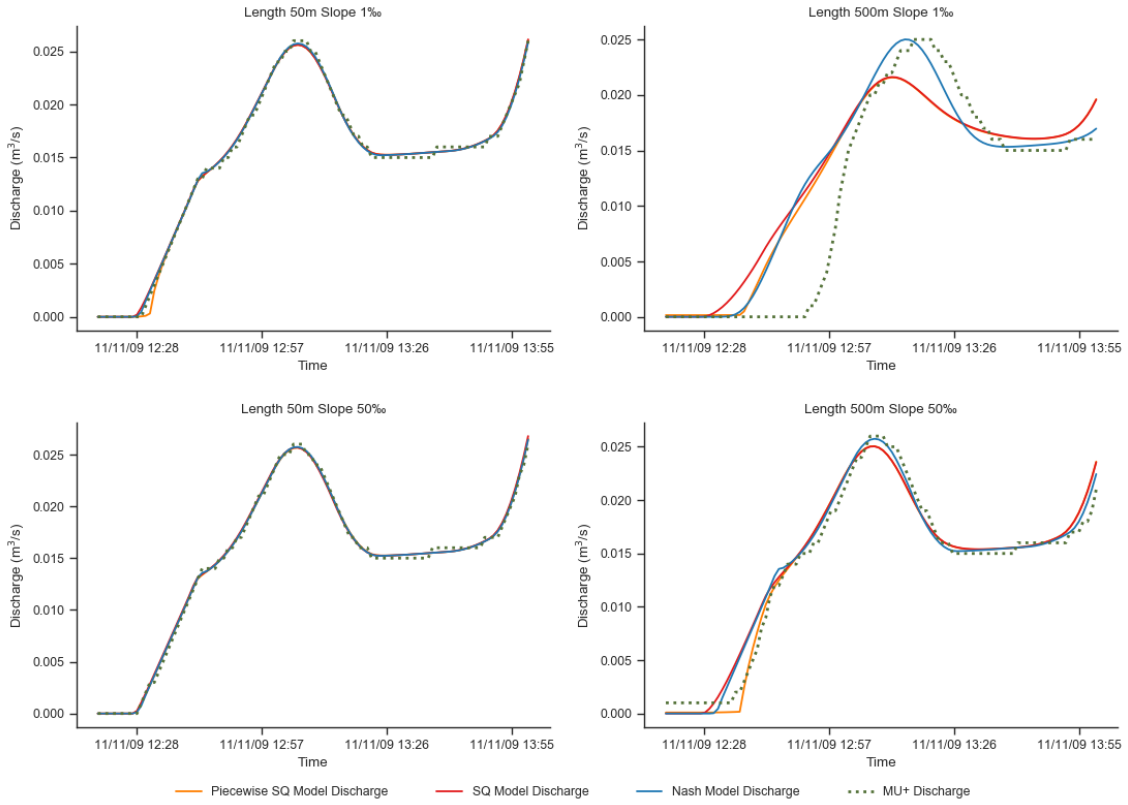

Figure G.3: Discharge comparison between MU+ and Nash model, single linear SQ model and piecewise linear SQ model

It can be seen that other than delaying the discharge further compared to the linear SQ function, the piecewise model does not change the magnitude of the peaks. The delay is also not so much. For the length $500m$ and slope $1‰$, the piecewise SQ model can only delay as much as the Nash model, both of which are still far from the time that discharge starts from MU+. This is unexpected because S_0 should still be delayed more, since the value of S_0 for this length and slope does not reach the maximum value tested yet. For other lengths and slopes, the extra delay seems to simulate quite well compared to MU+.

G.2.1.4 Issue of Increased Initial Storage Model

While examining the reason why the delay is not enough for length $500m$ and slope $1‰$, it was discovered that there is an issue with emptying water out of the transport stretch when initial storage is increased. Figure G.4 shows the behavior of volume of water in the pipe as an example. This phenomenon persists in all situation due to the formulation of the SQ relationship.

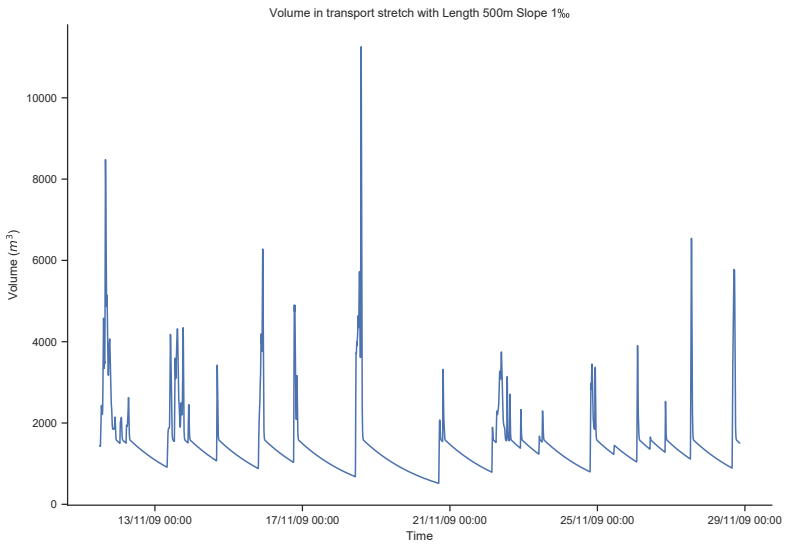

Figure G.4: Volume of water (m^3) inside the transport stretch length $500m$ slope $1‰$

From the figure, it is clear that once the storage inside the pipe reaches S_0 after the peaks, the pipe would not be emptied until a very long time later, and the storage inside the pipe only decreases slowly. This is because the SQ function used in this case delays the discharge by letting only $0.01\ m^3/s$ of water exit the transport stretch when the rain begins. However, the same process happens after the inflow reaches the peak. When the inflow decreases and the storage in the pipe reaches S_0, the discharge goes back to being only $0.01\ m^3/s$ again, making the system not able to become empty.

This scenario is worse compared to having a single linear SQ function. However, the optimization still chooses the piecewise function over the single linear model most of the time. The reason behind this is the way the optimization was done, by calculating RMSE excluding the dry periods. It means that the model is only superior compared to the linear SQ model in terms of delay at the beginning of the rain. The periods where the discharge cannot be emptied out of the system was not taken into consideration since these are the dry periods already neglected during optimization. Therefore, the piecewise SQ model cannot perform well in reality due to emptying issue.

G.2.2 City Simulations Using Increased Initial Storage Model

This section provides the results of the piecewise SQ function with increased initial storage while comparing it to the original case of having 1 single linear SQ function. All comparisons are conducted using only fine level of compartmentalization as a representation.

G.2.2.1 Star City

Figure G.5 illustrates the comparison of conceptual models using the linear and piecewise SQ functions for fine level of compartmentalization.

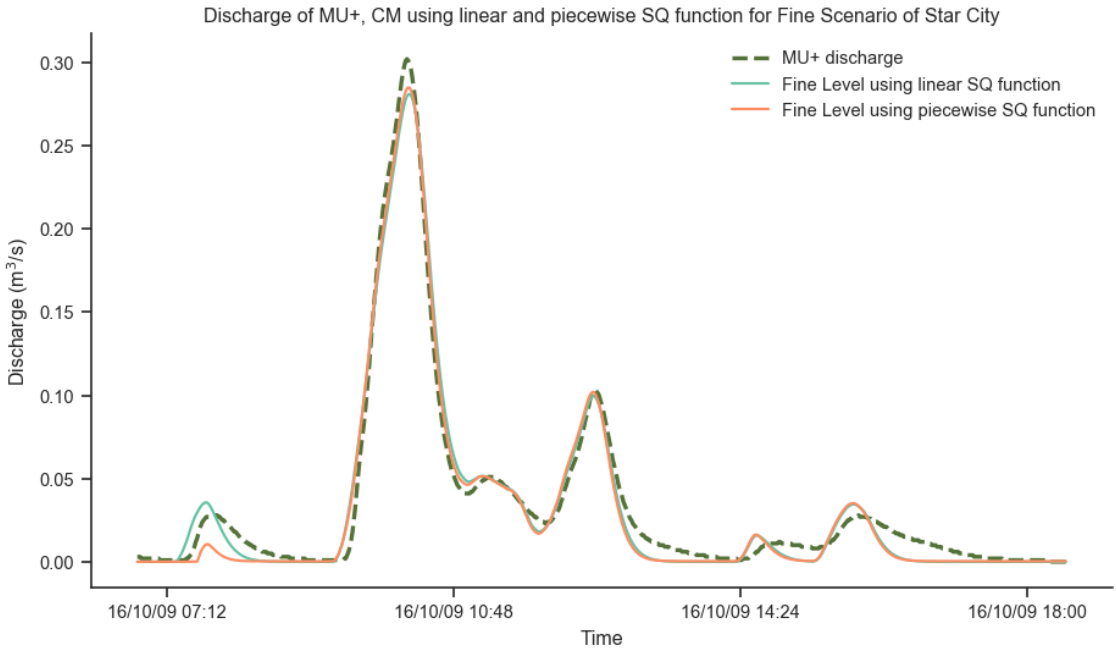

Figure G.5: Discharge comparison between MU+ and CM using piecewise and linear SQ functions for fine compartmentalization in Star city

The model works in terms of delaying the discharge during the first rain event. For this first inflow, the system starts discharging later but the peak still arrives slightly earlier compared to MU+. Peak magnitude is significantly smaller compared to the linear SQ model and MU+. For the rest of the rain, the difference between using piecewise or linear SQ models is not pronounced, as theoretically, their SQ slopes still remain more or less the same. The slight difference only occurs because the piecewise model directly optimizes the storage constant c, while the linear one optimizes it through the Nash model. Hence, errors and variations are introduced.

Figure G.6 shows the comparison between the conceptual model running with piecewise SQ function and MU+. In general, the difference with using linear SQ function (Figure 5.13) is rather insignificant. The model using piecewise SQ varies slightly less compared to MU+.

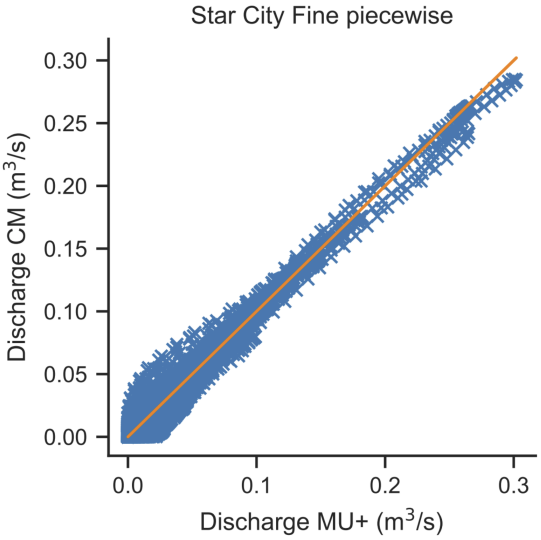

Figure G.6: Comparison between MU+ and CM discharges using piecewise and linear SQ functions for fine compartmentalization during 3 months of simulation in star city

In Figure G.7, it can be seen that the piecewise model actually performs worse than the linear SQ model when it comes to both magnitude and peak time difference in peaks compared to MU+. Not only having worse $PEP, PDIFF$ and $PTDIFF$, the piecewise SQ model also suffers more variation in peak simulations.

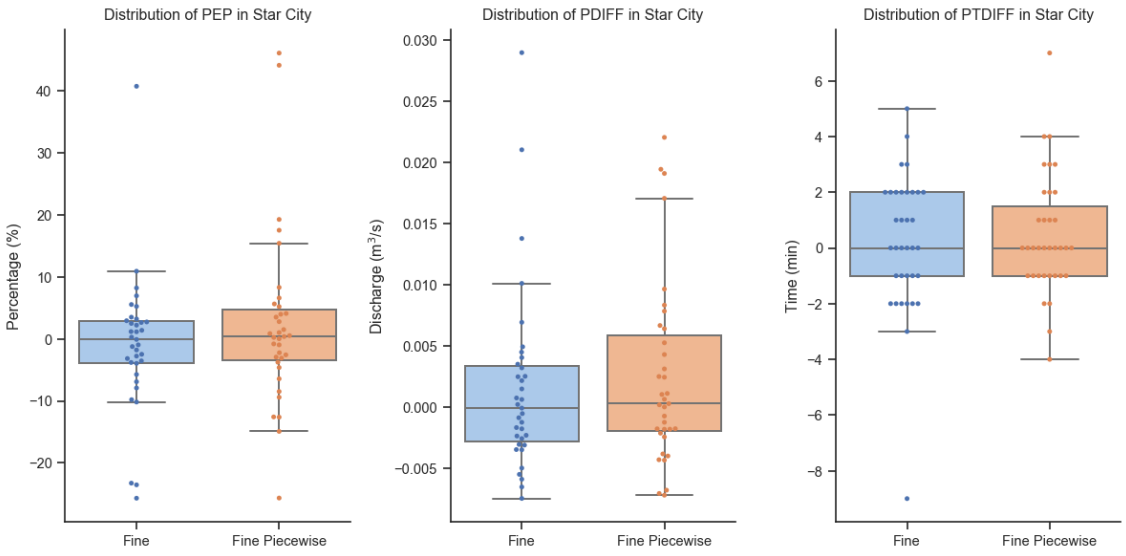

Figure G.7: $PEP, PDIFF$ and $PTDIFF$ between CM using piecewise and linear SQ functions for fine compartmentalization in Star city

The main difference compared to the linear SQ model is the value of PEP (Table G.1 and Table F.1). While the PEP for linear SQ model shows a negative value,

that of the piecewise SQ model has a positive value, meaning that this model has peaks that are more delayed compared to MU+. This is the result of having increased initial storage in the SQ function. The values of the model evaluation can be found in Table G.1.

Table G.1: Star city piecewise model evaluation for fine compartmentalization

NSE	PEP mean (%)	PEP standard error	PDIFF mean (m^3/s)	PDIFF standard error	PTDIFF mean (min)	PTDIFF standard error
0.9654	2.1782	2.3419	0.0025	0.0013	0.4286	0.3650

G.2.2.2 Strip City

For Strip city, the comparison between using different SQ function models can be seen in Figure G.8. Similar to the Star city, the piecewise model also manages to delay the discharge for the first rain event, but fails to simulate correctly the magnitude. Other than that, there is nearly no difference between using linear or piecewise model, as expected.

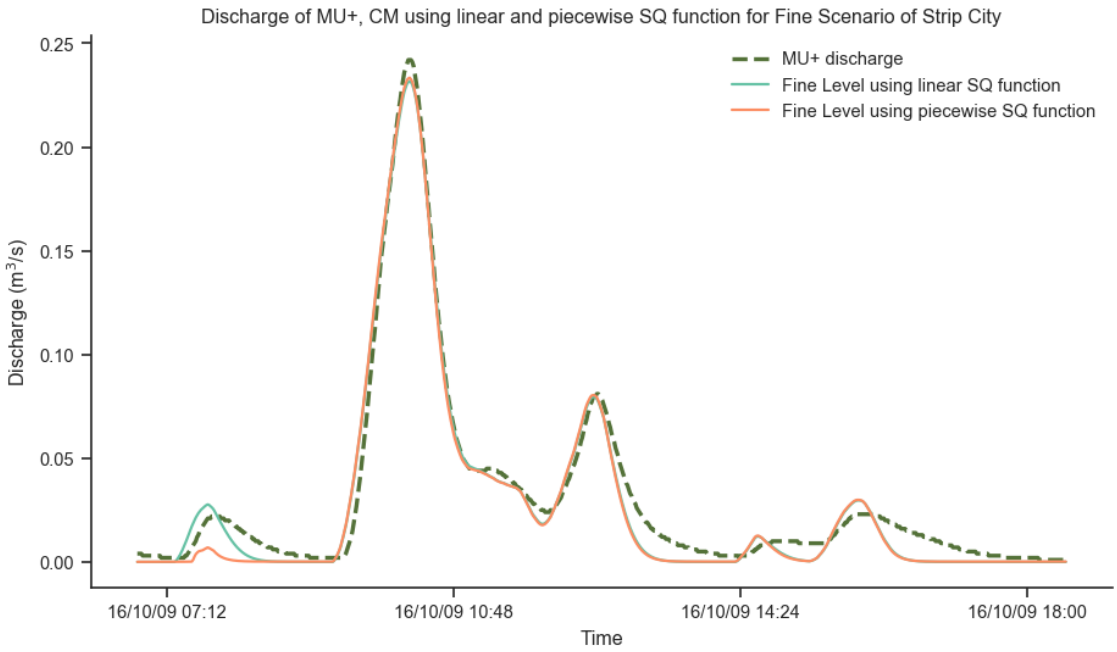

Figure G.8: Discharge comparison between MU+ and CM using piecewise and linear SQ functions for fine compartmentalization in Strip city

Figure G.9 illustrates the comparison of CM and MU+ in case of using piecewise model. Once again, the model performs similar to the linear SQ model (Figure 5.16).

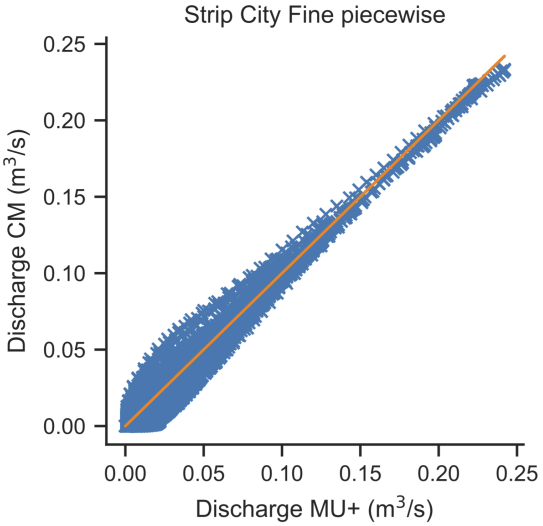

Figure G.9: Comparison between MU+ and CM discharges using piecewise and linear SQ functions for fine compartmentalization during 3 months of simulation in Strip city

Figure G.10 shows the performance of peak simulations between the two SQ models. Magnitude-wise, the piecewise model seems to improve marginally compared to the linear model, especially in PEP and $PDIFF$, with less variation. However, $PTDIFF$ between the two models are nearly the same.

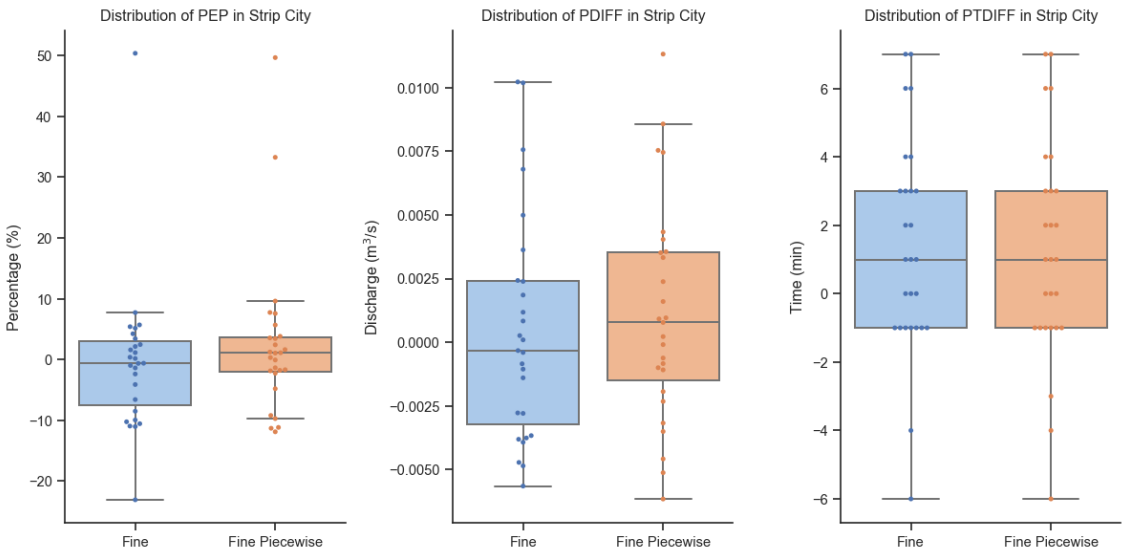

Figure G.10: PEP, $PDIFF$ and $PTDIFF$ between CM using piecewise and linear SQ functions for fine compartmentalization in Strip city

Table G.2 shows all values of peak evaluation as well as NSE for the piecewise model. On average, this model still performs worse than the linear model. However, NSE of the piecewise model is slightly higher than that of the linear one. The

behavior of PEP still follows that of Star city, meaning that the Strip city's PEP using piecewise model is now a positive value.

Table G.2: Strip city piecewise model evaluation

NSE	PEP mean (%)	PEP standard error	PDIFF mean (m^3/s)	PDIFF standard error	PTDIFF mean (min)	PTDIFF standard error
0.9520	2.3960	2.4833	0.0011	0.0008	1.2222	0.6209

G.3 Conclusion

From the performance of the piecewise SQ function using increased initial storage, it can be concluded that this model does not significantly alter the behavior of the discharge out of a city. Only the first rain event in the series of rain event is delayed more, but the magnitude is significantly lower than MU+. Not only does it not improve the performance of the conceptual models, the fact that after the peak of the rain, the pipes cannot be emptied quickly enough made this model not feasible to be used in simulation.

H Optimization of Single Linear SQ Model

The process of optimizing the single linear SQ model happens similar to that of the piecewise SQ function mentioned above (Appendix G). The only difference is that there is only 1 parameter to optimize, which is the slope of the SQ function $1/c$. This parameter is tested with values between 0.01 and 1 with the step of 0.01. The process of calculating errors to choose the best values still remains unchanged compared to Appendix G.

I Python Code

The code presented is only the one used for optimizing the parameters and simulating the cities. Plotting is omitted for convenience. This code is implemented in Python.

I.1 Loading Flows

This code is used to load the flow files from MU+. It converts in the correct unit and performs interpolation. There are various functions to help in different situations.

```
import os

import pandas as pd


def read_flow_series(location, timesteps_string="Time", values_string="Discharge outlet",
                     sampling_timesteps=60, date_format='%d-%m-%y %H:%M'):
    """
    Reads a generic flow series from MU+ with a given timestep, performing linear interpolation
    Params:
        - location: filepath for the MU+ file
        - timesteps_string: the name of the column of the timesteps
        - values_string: name of the column where the values are
        - sampling_timesteps: theperiod of each sampling timestep
        - date_format: the format of the date in the timestep column
    """
    dataframe = pd.read_csv(location)

    dataframe[timesteps_string] = pd.to_datetime(dataframe[timesteps_string], format=date_format)
    dataframe[values_string] = dataframe[values_string] * sampling_timesteps

    # select discharge data and remove empty rows
    dataframe = dataframe.dropna()
    # set discharge time as index of the table
    dataframe.set_index(pd.DatetimeIndex(dataframe[timesteps_string]), inplace=True)
    # resample to have every minute of data
    dataframe = dataframe.groupby(pd.PeriodIndex(data=dataframe[timesteps_string],
                                                 freq="{}S".format(
                                                     sampling_timesteps))).mean().resample(
        "{}S".format(sampling_timesteps)).asfreq()
    # reset index to be number instead of datetime (put datetime as column instead of index)
    dataframe.reset_index(inplace=True)
    dataframe[values_string] = dataframe[values_string].interpolate()

    return dataframe


def convert(filename):
    """
    Convert filename to length and slope parameters
    """
    split_list = filename[:-4].split("_")
    length = int(split_list[0][1:])
    slope = int(split_list[1][1:])
    return length, slope


def process_file(filepath, separator=";"):
    """
    Processes an inflow and discharge file from MU+
    """
```

```
    data_frame = pd.read_csv(filepath, sep=separator)

    data_frame['inflow time'] = pd.to_datetime(data_frame['inflow time'], format='%m/%d/%Y %H:%M')
    data_frame['inflow g7'] = data_frame['inflow g7'] * 60
    data_frame['discharge time'] = pd.to_datetime(data_frame['discharge time'],
                                                  format='%m/%d/%Y %H:%M')
    data_frame['discharge g7'] = data_frame['discharge g7'] * 60

    # select discharge data and remove empty rows
    discharge = data_frame[["discharge time", "discharge g7"]].copy().dropna()
    # set discharge time as index of the table
    discharge.set_index(pd.DatetimeIndex(discharge["discharge time"]), inplace=True)
    # resample to have every minute of data
    discharge = discharge.groupby(
        pd.PeriodIndex(data=discharge["discharge time"], freq="min")).mean().resample(
        'min').asfreq()
    # reset index to be number instead of datetime (put datetime as column instead of index)
    discharge.reset_index(inplace=True)
    discharge["discharge g7"] = discharge["discharge g7"].interpolate()

    inflow = data_frame[["inflow time", "inflow g7"]].copy().dropna()
    inflow.set_index(pd.DatetimeIndex(inflow["inflow time"]), inplace=True)
    inflow = inflow.groupby(pd.PeriodIndex(data=inflow["inflow time"], freq="min")).mean().resample(
        'min').asfreq()
    inflow.reset_index(inplace=True)
    inflow["inflow g7"] = inflow["inflow g7"].interpolate()

    return inflow, discharge


def load_file(folder, separator=";"):
    """
    Loads all the files in a folder and processes them as inflow and discharge files
    """
    filenames = os.listdir(folder)
    file_list = []
    for file in filenames:  # append only csv files into file_list
        if (file[-4:] == ".csv"):
            file_list.append(file)

    inflow_dict = {}
    discharge_dict = {}
    # load the processed inflow and discharge data, return inflow_dict and discharge_dict
    for file in file_list:
        filepath = folder + file
        inflow, discharge = process_file(filepath, separator)
        inflow_dict[file] = inflow
        discharge_dict[file] = discharge

    return inflow_dict, discharge_dict
```

I.2 Functions Used to Simulate and Optimize the Nash Model

This contains a collection of functions used to simulate the Nash model given k and n and optimize the parameters.

```
import load_files
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import os
from scipy.stats import gamma


def find_exclude(inflow):
    """
```

```
    Finds the timesteps to be excluded as they do not have any rain.
    """
    # to count number of cosecutive 0's
    counter = 0
    # tolerance to allow for some slack
    tolerance = 1e-3

    potential_exclude = []
    total_to_exclude = []

    # number of timesteps that are 0
    consecutive_zeros = 600

    timeseries = inflow["inflow g7"].to_numpy()  # convert pandas column into numpy array

    # timestep is index
    for timestep, value in enumerate(
            timeseries):  # enumerate gives a list made of index of element and the element as output

        # count consecutive 0, append timestep to potential_exclude
        if value < tolerance:
            counter += 1
            potential_exclude.append(timestep)
        else:
            if counter >= consecutive_zeros:
                # print(potential_exclude)
                total_to_exclude.append(np.array(potential_exclude))
            potential_exclude = []
            counter = 0

    total_to_exclude.append(potential_exclude)
    total_to_exclude = np.concatenate(total_to_exclude)

    return total_to_exclude


def unit_model_auto(n, k):
    """
    Creates unit hydrograph for 600 time steps for a combination of n and k. Automatically
    selects the correct number of timesteps to allow the interpolation to work.
    Params:
        - n: number of linear reservoirs
        - k: costant for each linear reservoir
    """
    #

    # find k s.t. P(x<k)>0.9999, to select most of the relevant values dynamically
    time_steps = max(10, gamma.isf(0.0001, n, 0.0, k))
    time = np.arange(time_steps)
    unit_hydro = gamma.cdf(time + 1, n, 0.0, k) - gamma.cdf(time, n, 0.0, k)
    return unit_hydro


def runoff_model_auto(inflow, n, k):
    """
    Finds the discharge by computing convolution of inflow and iuh of the Nash model.
        - inflow: the inflow timeseries
        - n: the number of linear reservoirs in the Nashmodel
        - k: the storage constant of the linear reservoirs.
    """
    outflow = np.convolve(unit_model_auto(n, k), inflow)
    outflow = outflow[0:len(inflow)]
    return outflow


def rms_error(outflow_series, result):
    """
    The root mean square error
    """
    error = np.sqrt(np.mean(np.power(outflow_series - result, 2)))
    return error
```

```
def find_best_k_n(file, ns, logks):
    """
    Finds the best parameters for the Nash model from the list of parameters
        - file: the filename of the inflow and discharge file
        - ns: the list of n's to be tested
        - logks: the list of log(k)'s to be tested
    """
    inflow, discharge = load_files.process_file("./Data/" + file)
    print("file loaded")

    excluded_timesteps = find_exclude(inflow)

    inflow_series = inflow["inflow g7"].to_numpy()
    outflow_series = discharge["discharge g7"].to_numpy()

    # select timesteps for error calculation, put False and unselect timesteps that are excluded
    mask = np.array([True] * len(inflow_series))
    mask[excluded_timesteps] = False

    length, slope = load_files.convert(file)
    best_error = np.inf
    # create empty table of 0 that has number of rows = no of n values, no of columns = no of logks values
    errors = np.zeros((len(ns), len(logks)))
    i = 0

    for n in ns:
        j = 0
        for x in logks:
            result = runoff_model_auto(inflow_series, n, math.pow(10, x))
            error = rms_error(outflow_series[mask], result[mask])
            if error < best_error:
                best_error = error
                best_n = n
                best_k = math.pow(10, x)
            errors[i, j] = error
            j += 1
        i += 1
    best_n_k = {
        "file": file,
        "best n": best_n,
        "best k": best_k,
        "slope": slope,
        "length": length
    }

    np.save("errors_{}.npy".format(file[:-4]), errors)

    return best_n_k, errors
```

I.3 Solve differential equations of the Nash model

In this file, the differential equations of the Nash model are defined and solved to obtain the SQ relationship for the given pipe. Then, the values of the SQ plot are interpolated to find the storage coefficient of each pipe.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint


def solve_diff_eq(inflow, length, slope, n, k, nstep, store=True, folder="SQ_results"):
    """
    Solve the differential equation for given k and n parameters of the Nash model.
    Params:
        - inflow: inflow to be used in the solution of the diff eq
        - length: length of the pipe (used for display)
        - slope: slope of the pipe (used for display)
        - n: number of linear reservoirs
```

```
        - k: storage constant of each linear reservoir
        - nstep: number of steps to be simulated
        - store: whether to store the results
        - folder: where to store the results
    """
    times = np.arange(nstep)

    # function that returns dz/dt
    def model(z, t, k, n, inflow):
        ik = 1 / k
        dzdt = [inflow(t) - z[0] * ik]
        for i in range(1, n):
            dzdt.append((z[i - 1] - z[i]) * ik)

        return dzdt

    # initial condition
    z0 = np.zeros((n,))

    # solve ODE
    z = odeint(model, z0, times, args=(k, n, inflow))
    # plot s versus q
    s = np.sum(z, 1)
    q = (z[:, -1] / k) / 60
    plt.figure()
    plt.plot(s, q, "x")
    plt.title("SQ function {} {}".format(length, slope))
    plt.xlabel("Storage $m^3$")
    plt.ylabel("Discharge $m^3/s$")
    plt.show()
    if store:
        np.save(folder + "/L{}_S{}.npy".format(length, slope), z)
    return z


def find_slope_simulate_single_pipe(length, slope, inflow, nstep, store=False):
    """
    Simulates a single pipe and finds the slope of the SQ function
    """
    error_file = "errors_L" + str(length) + "_S" + str(slope) + ".npy"

    error_table = merged_errors[error_file]

    best_n_id, best_x_id = np.unravel_index(np.argmin(error_table), error_table.shape)

    best_n, best_k = ns[best_n_id], np.power(10, logks[best_x_id])

    # run storage simulation
    z = solve_diff_eq(inflow, length, slope, int(best_n), best_k, nstep, store, folder)

    s = np.sum(z, 1)
    q = (z[:, -1] / best_k) / 60
    regression_slope = np.sum(q) / np.sum(s)
    results = {
        "length": int(length),
        "slope": int(slope),
        "big_k": 1 / (regression_slope),
        "k": best_k,
        "k_secs": best_k * 60,
        "n": best_n
    }
    plt.figure()
    plt.plot(s, q, "x", label="Original data")
    plt.plot(s, regression_slope * s, 'r', label='fitted line')
    plt.title("L{} S{}".format(length, slope))
    plt.xlabel("Storage $m^3$")
    plt.ylabel("Discharge $m^3/s$")
    plt.show()
    plt.close()

    return results
```

I.4 Conceptual Model Library

This is a library of functions used to run the conceptual models. It contains functions to manage the city network, find the SQ function given the specified parameters, create the PRM file, simulate a single transport stretch using the DtuSmModels engine and simulate an entire city.

```
import os
import sys
import clr
import numpy as np
import pandas as pd

sys.path.append(os.path.relpath('./Release'))
clr.AddReference('DtuSmModels')
from DtuSmModels import *
import cloudpickle


def get_SQ_string_general(slopes, storage_steps, q0, qmax):
    """
    Generates the SQ string used in the connection described in the PRM file.
    Params:
        - slopes: list of slopes of the SQ plot.
        - storage_steps: list of storage steps. The first slope starts after the first storage step.
        - qmax: maximum flow for the SQ plot.
    """
    if (storage_steps[0] == 0):
        SQ_function = [(0, 0)]
    else:
        SQ_function = [(0, 0), (round(storage_steps[0], 2), round(q0, 2))]

    last_q = q0
    last_s = storage_steps[0]

    assert len(storage_steps) == len(slopes)

    # we already used the first one
    for i, storage in enumerate(storage_steps[1:]):
        next_q = last_q + (slopes[i] * (storage - last_s))
        next_s = storage
        SQ_function.append((round(next_s, 2), round(next_q, 2)))
        last_s = next_s
        last_q = next_q
    s_max = (qmax - last_q) / slopes[-1] + last_s
    SQ_function.append((round(s_max, 2), round(qmax, 2)))
    SQ_function.append((round(s_max * 500, 2), round(qmax * 1.1, 2)))
    SQ_string = ";".join([str(x[0]) + "," + str(x[1]) for x in SQ_function])
    SQ_string = "(" + SQ_string + ")"

    # SQ_string = "(0,0;"+str(Smax)+","+str(qmax)+";"+str(Smax*500)+','+str(qmax*1.1)+")"

    return SQ_string


def get_SQ_string(slope, qmax, function_type="linear"):
    """
    Generates the SQ string used in the connection described in the PRM file
    Params:
        - slope: slope of the SQ plot
        - qmax: maximum flow for the SQ plot
        - function_type: SQ function type, for now only linear is supported
    """
    if function_type == "linear":
        if pd.isna(slope):
            raise Exception("Slope is a Null value")
        Smax = round((qmax) / slope, 2)  # storage value corresponding to qmax in the S-Q-function
        # create an S_Q function that corresponds to the above parameters
        SQ_string = "(0,0;" + str(Smax) + "," + str(qmax) + ";" + str(Smax * 500) + ',' + str(
```

```
            qmax * 1.1) + ")"

    return SQ_string


def create_connections(from_name, to_names, SQ_strings, connection_types=None):
    """
    Creates the text required to add a "pipe" in the PRM file. Supports multiple connections
    Params:
        - from_name: name of the node where the connection starts
        - to_names: list of the names where each connection ends in
        - connection_types: the type of SQ function used in each connection
        - SQ_string: the string describing the SQ function used in each connection
    """
    # convert to array if only one connection is provided
    if isinstance(to_names, str):
        to_names = [to_names]
    if isinstance(SQ_strings, str):
        SQ_strings = [SQ_strings]

    if connection_types is None:
        connection_types = ["PieceWiseLinRes"] * len(SQ_strings)

    # check that everything has the same length
    assert (len(to_names) == len(connection_types)) and (len(to_names) == len(SQ_strings))

    string = "\t**********************************\n"
    string += "\t<name> {}\n".format(from_name)
    string += "\t<type>   drainage\n"
    for i in range(len(to_names)):
        string += "\t<connection> {} {} {}\n".format(to_names[i], connection_types[i],
                                                     SQ_strings[i])

    return string


def create_surface_model(name, surface_type="TA1", imperv_area=60000):
    """
    Creates the text for the surface model
    Params:
        - name: name of the node having this surface model.
        - surface_type: type of surface for the node.
    """
    avail_surface_types = ["TA1"]
    if surface_type == "TA1":
        return "\t<SurfMod> {} {} ({},1)\n".format(name, surface_type, imperv_area)
    else:
        print("Surface type not recognized: {}.".format(surface_type))
        print("Available surface types are: {}.".format(avail_surface_types))
        return ""


def poly_transform(data, power=1):
    """
    Transofrm data for early experimental extrapolation.
    """
    a = np.ones((data.shape[0], 1))
    a = np.concatenate((a, data), axis=1)

    for i in range(2, power + 1):
        a = np.concatenate((a, np.power(data, i)), axis=1)
    a = np.concatenate((a, np.expand_dims(np.log(data[:, 0]), 1)), axis=1)
    return a


def find_slope_SQ_plot(length, slope, SQ_slopes, method="closest"):
    """
    Finds the values of the slope of the SQ plot (in 1/s) given the length and slope of a pipe
    Params:
        - length: length of the pipe
        - slope: slope of the pipe
        - SQ_slopes: dictionary with the slope values and respective length and slope of pipes
```

```
        - method: the way in which the SQ slope is computed. Supported: interpolate, closest, model.
    """

    if method == "closest":
        # use dataframe to find all the lengths and slopes available
        available_lengths = pd.unique(SQ_slopes["length"])
        available_slopes = pd.unique(SQ_slopes["slope"])

        # find closest matching length and slope separately
        closest_length = available_lengths[np.argmin(np.abs(length - available_lengths))]
        closest_slope = available_slopes[np.argmin(np.abs(slope - available_slopes))]

        # return the SQ slope in seconds
        return SQ_slopes.loc[closest_slope, closest_length]["SQ_slope"] / 60
    elif method == "interpolate":
        # use dataframe to find all the lengths and slopes available
        available_lengths = pd.unique(SQ_slopes["length"])
        available_slopes = pd.unique(SQ_slopes["slope"])

        # if withing range, then use closest method, it's more accurate
        if length <= available_lengths[-1]:
            return find_slope_SQ_plot(length, slope, SQ_slopes, "closest")

        # use interpolation
        power = 1
        linear_a = SQ_slopes[["length", "slope"]].to_numpy()
        a = poly_transform(linear_a, power)
        b = SQ_slopes["SQ_slope"].to_numpy()
        x, residuals, rank, s = np.linalg.lstsq(a, b, rcond=None)

        # return the SQ slope in seconds
        return np.dot(poly_transform(np.array([[length, slope]]), power), x)[0] / 60
    elif method == "model":
        # use new model, for exepriments with different interpolations
        available_lengths = pd.unique(SQ_slopes["length"])
        available_slopes = pd.unique(SQ_slopes["slope"])

        # if withing range, then use closest method, it's more accurate
        if length <= available_lengths[-1]:
            return find_slope_SQ_plot(length, slope, SQ_slopes, "closest")

        # use the model trained to be good at extrapolation
        file = open('model.joblib', "rb")
        clf = cloudpickle.load(file)

        # return the SQ slope in seconds
        return clf.predict(np.array([[length, slope]]))[0] / 60


def find_params_SQ_plot(length, slope, dataframe, method="closest"):
    """
    Finds the parameters of the SQ plot given a dataframe and a length and slope. This finds
    the entire set of parameters that are available in the dataframe. The columns should contain
    "length", "slope" and then all the other parameters
    Params:
        - length: length of the pipe
        - slope: slope of the pipe
        - SQ_slopes: dictionary with the parameters with respective length and slope of pipes
        - method: the way in which the SQ slope is computed. Supported: closest.
    """
    params = {}
    if method == "closest":
        available_lengths = pd.unique(dataframe["length"])
        available_slopes = pd.unique(dataframe["slope"])

        closest_length = available_lengths[np.argmin(np.abs(length - available_lengths))]
        closest_slope = available_slopes[np.argmin(np.abs(slope - available_slopes))]

        available_param_names = list(set(dataframe.columns) - set(["length", "slope"]))
        for name in available_param_names:
            params[name] = dataframe.loc[
                (dataframe["slope"] == closest_slope) & (
```

```
                        dataframe["length"] == closest_length), name].to_numpy()[0]
    return params


def find_qmax(diameter, slope, manning_coeff=1 / 85):
    """
    Finds the maximum flow through a pipe using the Manning Equation.
    - diameter: diameter of the pipe (meters).
    - slope: slope of the pipe.
    - manning_coeff: manning_coefficient.
    """
    A_v = np.pi * ((diameter / 2.) ** 2)  # pipe cut area, sqm
    Q_v = 1 / manning_coeff * A_v * (
            np.power((diameter / 4.0), (2.0 / 3.0)) * (np.sqrt(slope / 1000)))

    return Q_v


def process_inflow(filepath, separator=";", timestep=60):
    """
    Processes the inflow file into each catchment for the city simulation.
    Params:
        -  filepath: path to the file of the inflow
        -  separator: separator of the csv file
        -  timestep: length of the timestep for the sampling
    """
    data_frame = pd.read_csv(filepath, sep=separator)

    data_frame['inflow time'] = pd.to_datetime(data_frame['inflow time'], format='%m/%d/%Y %H:%M')
    data_frame['inflow g7'] = data_frame['inflow g7'] * timestep
    data_frame['discharge time'] = pd.to_datetime(data_frame['discharge time'],
                                                  format='%m/%d/%Y %H:%M')
    data_frame['discharge g7'] = data_frame['discharge g7'] * timestep

    # select discharge data and remove empty rows
    discharge = data_frame[["discharge time", "discharge g7"]].copy().dropna()
    # set discharge time as index of the table
    discharge.set_index(pd.DatetimeIndex(discharge["discharge time"]), inplace=True)
    # resample to have every minute of data
    discharge = discharge.groupby(
        pd.PeriodIndex(data=discharge["discharge time"], freq="min")).mean().resample(
        str(timestep) + 'S').asfreq()
    # reset index to be number instead of datetime (put datetime as column instead of index)
    discharge.reset_index(inplace=True)
    discharge["discharge g7"] = discharge["discharge g7"].interpolate("linear")

    inflow = data_frame[["inflow time", "inflow g7"]].copy().dropna()
    inflow.set_index(pd.DatetimeIndex(inflow["inflow time"]), inplace=True)
    inflow = inflow.groupby(pd.PeriodIndex(data=inflow["inflow time"], freq="min")).mean() \
                .resample(str(timestep) + 'S').asfreq()
    inflow.reset_index(inplace=True)
    inflow["inflow g7"] = inflow["inflow g7"].interpolate("linear")

    return inflow, discharge


import itertools


# graphs implemented taking inspiration from https://www.python.org/doc/essays/graphs/
def get_subnetwork_nodes(graph, nodes):
    all_nodes = list(graph.keys())
    subnetwork = {}
    for node in graph.keys():
        if node in nodes:
            subnetwork[node] = [x for x in graph[node] if x in nodes]

    return subnetwork


def find_shortest_path(graph, start, end, path=[]):
    path = path + [start]
```

```
    if start == end:
        return path
    if start not in graph:
        return None
    shortest = None
    for node in graph[start]:
        if node not in path:
            newpath = find_shortest_path(graph, node, end, path)
            if newpath:
                if not shortest or len(newpath) < len(shortest):
                    shortest = newpath

    return shortest


def find_longest_path(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if start not in graph:
        return None
    longest = None
    for node in graph[start]:
        if node not in path:
            newpath = find_longest_path(graph, node, end, path)
            if newpath:
                if not longest or len(newpath) > len(longest):
                    longest = newpath

    return longest


def find_longest_path_network(graph):
    # checks through all the nodes in network to find the longest path. Only uses number of pipes
    # and not length of the pipes (speeds up the process and is approximately correct for this case)
    longest = None
    for node1, node2 in itertools.combinations(list(graph.keys()), 2):
        current = find_longest_path(graph, node1, node2)
        if current:
            if not longest or len(current) > len(longest):
                longest = current
        current = find_longest_path(graph, node2, node1)
        if current:
            if not longest or len(current) > len(longest):
                longest = current
    return longest


def find_all_pipes_network(graph):
    pipes = []
    for start_node in graph.keys():
        for end_node in graph[start_node]:
            pipes.append("{}-{}".format(start_node, end_node))

    return pipes


def load_runoff(filepath, separator=","):
    """
    Loads the runoff file and puts it in the usual format to be used (m^3/min) and sampling every
    minute.
    Params:
        - filepath: location of the inflow data csv file.
        - separator: separator used in the csv file.
    """
    data_frame = pd.read_csv(filepath, sep=separator)

    data_frame['Time'] = pd.to_datetime(data_frame['Time'], format='%m/%d/%Y %H:%M')
    data_frame['Runoff'] = data_frame['Runoff'] * 60

    # select discharge data and remove empty rows
    data_frame = data_frame.dropna()
```

```
    # set discharge time as index of the table
    data_frame.set_index(pd.DatetimeIndex(data_frame["Time"]), inplace=True)
    # resample to have every minute of data
    data_frame = data_frame.groupby(
        pd.PeriodIndex(data=data_frame["Time"], freq="min")).mean().resample('min').asfreq()
    # reset index to be number instead of datetime (put datetime as column instead of index)
    data_frame.reset_index(inplace=True)
    data_frame["Runoff"] = data_frame["Runoff"].interpolate()

    return data_frame


def merge_runoff_network(graph, folder="star_city_data", prefix="star_city_catchment_runoff_"):
    """
    Sums up all the inflow for a compartment, looking through the files in "folder" that start with
    "prefix"
    Params:
        - graph: the compartment.
        - folder: the folder where the inflows are stored.
        - prefix: the prefix used for all the inflow files.
    """
    runoffs = []
    cumulative_runoff = None
    for node in graph.keys():
        if len(graph[node]) == 0:
            continue
        file = os.path.join(folder, "{}{}.csv".format(prefix, node))
        if os.path.exists(os.path.join(folder, "{}{}.csv".format(prefix, node))):
            # check this because for some nodes there is no inflow
            runoffs.append(load_runoff(file))
            if cumulative_runoff is None:
                timesteps = runoffs[-1]["Time"]
                cumulative_runoff = runoffs[-1]["Runoff"]
            else:
                cumulative_runoff += runoffs[-1]["Runoff"]
    if cumulative_runoff is None:
        return None
    else:
        return pd.concat([timesteps, cumulative_runoff], axis=1)


def total_area_compartment_with_network(network, dataframe_info):
    """
    Shortcut for total_area_compartment()
    """
    pipes = find_all_pipes_network(network)
    data = dataframe_info[dataframe_info["Pipe"].isin(pipes)][["Pipe", "Total area "]]
    total_area = 0
    for pipe in pipes:
        cur_pipe = data[data["Pipe"] == pipe]
        total_area += cur_pipe["Total area "].to_numpy()[0]
    return total_area


def total_area_compartment(city_network, nodes_in_comp, comp_name, dataframe_info):
    """
    Finds total area of the compartment.
    """
    network = get_subnetwork_nodes(city_network, nodes_in_comp[comp_name])
    return total_area_compartment_with_network(network, dataframe_info)


def get_length_slope_diameter_compartment(compartment_net, dataframe_info, length_mode="wavg",
                                          slope_mode="wavg", diameter_mode="avg",
                                          forced_length=None, forced_slope=None,
                                          forced_diameter=None):
    """
    Finds the parameters for the compartment based on the choices.
    Params:
        - comparment_net: the network of the compartment.
        - dataframe_info: the dataframe containing the information necessary to build the city
                          (coming from the rational method).
```

```
    """
    # find all the pipes in the network
    pipes = find_all_pipes_network(compartment_net)
    # read only useful information
    data = dataframe_info[dataframe_info["Pipe"].isin(pipes)][
        ["Pipe", "Total area ", "Chosen slope, Ib", "Pipe length, L", "Dimension, d"]]
    diameter = 0
    slope = 0
    length = 0
    w_diam = 0
    w_slope = 0
    w_length = 0
    for pipe in pipes:
        cur_pipe = data[data["Pipe"] == pipe]
        weight = cur_pipe["Total area "].to_numpy()[0]
        weight = weight if weight > 0 else 1

        # compute diameter
        if diameter_mode == "wavg":
            diameter += weight * cur_pipe["Dimension, d"].to_numpy()[0]
            w_diam += weight
        elif diameter_mode == "avg":
            diameter += cur_pipe["Dimension, d"].to_numpy()[0]
            w_diam += 1
        elif diameter_mode == "max":
            diameter = np.maximum(diameter, cur_pipe["Dimension, d"].to_numpy()[0])
        else:
            print("diameter mode not recognized, it was: {}. Defaulting to max.".format(
                diameter_mode))
            diameter = np.maximum(diameter, cur_pipe["Dimension, d"].to_numpy()[0])

        # compute slope
        if slope_mode == "wavg":
            slope += weight * cur_pipe["Chosen slope, Ib"].to_numpy()[0]
            w_slope += weight
        elif slope_mode == "avg":
            slope += cur_pipe["Chosen slope, Ib"].to_numpy()[0]
            w_slope += 1
        elif slope_mode == "min":
            slope = np.minimum(slope, cur_pipe["Chosen slope, Ib"].to_numpy()[0])
        else:
            print("slope mode not recognized, it was: {}. Defaulting to wavg.".format(slope_mode))
            slope += weight * cur_pipe["Chosen slope, Ib"].to_numpy()[0]
            w_slope += weight

        # compute length of pipe
        if length_mode == "wavg":
            length += weight * cur_pipe["Pipe length, L"].to_numpy()[0]
            w_length += weight
        elif length_mode == "avg":
            length += cur_pipe["Pipe length, L"].to_numpy()[0]
            w_length += 1

    if (length_mode == "wavg") or (length_mode == "avg"):
        length /= w_length

    if (diameter_mode == "wavg") or (diameter_mode == "avg"):
        diameter /= w_diam

    if (slope_mode == "wavg") or (slope_mode == "avg"):
        slope /= w_slope

    # special case for the longest sum
    if length_mode == "lsum":
        length = 0
        path = find_longest_path_network(compartment_net)
        pipes_in_path = [path[i] + "-" + path[i + 1] for i in range(len(path) - 1)]
        for pipe in pipes_in_path:
            cur_pipe = data[data["Pipe"] == pipe]
            length += cur_pipe["Pipe length, L"].to_numpy()[0]

    # replace everything is a value of length and slope was asked
```

```
    if forced_length:
        length = forced_length
    if forced_slope:
        slope = forced_slope
    if forced_diameter:
        diameter = forced_diameter

    # scale the diameter to be in meters
    return length, slope, diameter / 1000


def create_PRM_file_city(city_network, comp_network, nodes_in_comp, SQ_df_info, info_city,
                         filepath="", city_name="", length_mode="wavg", slope_mode="wavg",
                         diameter_mode="max", SQ_slope_mode="model", forced_slope=None,
                         forced_length=None, forced_diameter=None, use_piecewise_SQ=False,
                         scale_SQ_slope=1):
    """
    Creates the PRM file (returns its filepath).
    """
    # start the hydraulic model
    PRM_file = "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n" \
               "[HydraulicModel]\n" \
               "\t**********************************\n" \
               "\t<name> outlet\n" \
               "\t<type>   drainage\n"

    nodes = set()

    comp_info = {}

    for edge in comp_network:
        from_name = edge[0]
        to_name = edge[1]

        # add to the set of nodes, create surface models based on this later
        nodes.add(from_name)
        nodes.add(to_name)

        network = get_subnetwork_nodes(city_network, nodes_in_comp[from_name])

        length, \
        slope, \
        diameter = get_length_slope_diameter_compartment(network,
                                                         info_city,
                                                         length_mode=length_mode,
                                                         slope_mode=slope_mode,
                                                         diameter_mode=diameter_mode,
                                                         forced_length=forced_length,
                                                         forced_slope=forced_slope,
                                                         forced_diameter=forced_diameter)
        if use_piecewise_SQ:
            # find all the parameters for the SQ function of this pipe
            params = find_params_SQ_plot(length, slope, SQ_df_info, method="closest")

            s0 = params["SQ_storage_0"]
            slope0 = scale_SQ_slope * params["SQ_slope_0"] / 60
            q0 = params["SQ_q_0"]
        else:
            # find the slope for the SQ function of this pipe
            SQ_slope = find_slope_SQ_plot(length, slope, SQ_df_info, method=SQ_slope_mode)

            s0 = 0
            slope0 = SQ_slope * scale_SQ_slope
            q0 = 0

        # get the SQ string describing the SQ function TODO: define different q_max for each pipe?
        SQ_string = get_SQ_string_general([slope0], [s0], q0, find_qmax(diameter, slope) * 60)
        comp_info[from_name] = {
            "SQ_string": SQ_string,
            "s0": s0,
            "slope0": slope0,
            "q0": q0,
```

```
            "qmax": find_qmax(diameter, slope) * 60,
            "diameter": diameter,
            "slope": slope,
            "length": length,
            "num_pipes": len(find_all_pipes_network(network)),
            "total_area": total_area_compartment_with_network(network, info_city),
            "num_manholes": len(network.keys())
        }
        print(comp_info[from_name])

        # add the pipe to the PRM file
        PRM_file += create_connections(from_name, to_name, SQ_string)

    # end the description of the hydaulic model
    PRM_file += "[EndSect]\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n" \
                "\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n"

    # start runoff information
    PRM_file += "[Runoff]\n" \
                "[SurfaceModels]\n"

    for node in nodes:
        if node == "outlet":
            continue

        if total_area_compartment(city_network, nodes_in_comp, node, info_city) == 0:
            continue

        PRM_file += create_surface_model(node)

    PRM_file += "[EndSect]\n" \
                "[EndSect]\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n"

    SmParameterFile = os.path.abspath(
        r'{}'.format(os.path.join(filepath, "PRM_FILE_{}.prm".format(city_name))))
    f = open(SmParameterFile, 'w')
    f.writelines(PRM_file)
    f.close()

    return SmParameterFile, comp_info


def run_model(prm_file_path, city_network, comp_network, nodes_in_comp, info_city,
              runoff_folder="star_city_data", runoff_prefix="star_city_catchment_runoff_"):
    """
    Runs the model defined in a PRM file using the DtuSmModel engine.
        - prm_file_path: path to the PRM file describing the network
        - city_network: the city network graph
        - comp_network: the compartment network
        - nodes_in_comp: the nodes in each compartment
        - info_city: the dataframe with information on the city (from rational method)
        - runoff_folder: the folder where the runoff files are located
        - runoff_prefix: the prefix of the runoff files
    """
    model = MainModel()
    model.initializeFromFile(prm_file_path)

    for edge in comp_network:
        # find name of starting node and ending node
        from_name = edge[0]
        to_name = edge[1]
        # define outputs that will be saved by the model
        # FROM, TO, NAME_OF_THE_OUTPUT_VARIABLE
        model.addOutputVariable(from_name, to_name, "{}-{}".format(from_name, to_name))

    for edge in comp_network:
        # find name of starting node and ending node
        from_name = edge[0]
```

```python
        if total_area_compartment(city_network, nodes_in_comp, from_name, info_city) == 0:
            continue

        network = get_subnetwork_nodes(city_network, nodes_in_comp[from_name])
        # get the inflow series
        runoff = merge_runoff_network(network, folder=runoff_folder, prefix=runoff_prefix)
        inflowseries = runoff["Runoff"].to_numpy()

        model.setIndividualRainData(from_name, inflowseries)

    model.runForOneMinuteRainInput()

    outputs = {}
    for i in range(model.output.getData().Length):
        outputs[model.output.dataCollection[i].name] = model.output.dataCollection[i].data

    timevector = model.output.timeInSeconds

    statevector = []
    for i in range(model.state.values.Length):
        statevector.append(model.state.values[i])

    return outputs, timevector, statevector


def simulate_pipe_fast(inflow, SQ_string, name, folder="PRM_single_pipe_new"):
    """
        Simulates a pipe with a certain inflow using the DtuSmModel engine.
        Params:
            - inflow: inflow for the pipe
            - SQ_string: string of the SQ function describing how the pipe behaves
            - name: name of the simulation, used for storage
            - folder: folder where to store the PRM files
        """
    if not os.path.exists(folder):
        os.makedirs(folder)

    # start the hydraulic model
    PRM_file = "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n" \
               "[HydraulicModel]\n" \
               "\t************************************\n" \
               "\t<name> outlet\n" \
               "\t<type>   drainage\n"

    # add the pipe to the PRM file
    PRM_file += create_connections("A1", "outlet", SQ_string)

    # end the description of the hydaulic model
    PRM_file += "[EndSect]\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n" \
                "\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n"

    # start runoff information
    PRM_file += "[Runoff]\n" \
                "[SurfaceModels]\n"

    PRM_file += create_surface_model("A1")

    PRM_file += "[EndSect]\n" \
                "[EndSect]\n" \
                "/>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n"

    SmParameterFile = os.path.abspath(os.path.join(folder, 'PRM_AUTO_{}.prm'.format(name)))
    f = open(SmParameterFile, 'w')
    f.writelines(PRM_file)
    f.close()
    model = MainModel()
    model.initializeFromFile(SmParameterFile)

    model.addOutputVariable("A1", "outlet", "A1-outlet")
```

```
    timestep = 60

    model.setIndividualRainData("A1", inflow)

    model.runForOneMinuteRainInput()
    outputs = {}
    for i in range(model.output.getData().Length):
        outputs[model.output.dataCollection[i].name] = model.output.dataCollection[i].data

    return np.array(outputs["A1-outlet"])
```

I.5 SQ Function Optimization

This code is used to find the optimal parameters for the SQ function of a transport stretch. Its discharge is simulated and compared to the one from MU+. The code used is the same for both linear SQ function and piecewise SQ function, what is changed in the different scenarios are the parameters optimized. Below, the stepwise configuration is shown, which is the most complicated one.

```
import os
import sys
from datetime import datetime
from multiprocessing import Process, Pool, Lock

import clr
import numpy as np

import k_n_functions
import load_files
import conceptual_model_lib

sys.path.append(os.path.relpath('./Release'))
clr.AddReference('DtuSmModels')
from DtuSmModels import *

qmax = 200

s0_start = 0
s0_end = 400
s0_step = 25

slope0_start = 0.01
slope0_end = 1.
slope0_step = 0.01

q0_start = 0.01
q0_end = 0.1
q0_step = 0.09

folder = "stepwise-s0_{}_{}_{}-slope0_{}_{}_{}-q0_{}_{}_{}".format(s0_start, s0_step, s0_end,
                                                                   slope0_start, slope0_step,
                                                                   slope0_end, q0_start, q0_step,
                                                                   q0_end)
if not os.path.isdir(folder):
    os.mkdir(folder)

s0s = np.arange(s0_start, s0_end + s0_step, s0_step)
print("S0's tested:", s0s)
slope0s = np.arange(slope0_start, slope0_end + slope0_step, slope0_step)
print("slope's tested:", slope0s)
q0s = np.arange(q0_start, q0_end + q0_step, q0_step)
print("q0's tested:", q0s)
print("Total number of parameters tested:", len(s0s) * len(slope0s) * len(q0s))


def find_best_slope_linear(params):
    # parse input parameters
```

```
    file = params["file"]
    s0s = params["s0s"]
    q0s = params["q0s"]
    slope0s = params["slope0s"]
    data_folder = params.get("data_folder", "data_new")
    output_folder = params.get("output_folder", "experiment_linearSQ")

    inflow, discharge = load_files.process_file(os.path.join(data_folder, file), separator=",")
    print("file loaded")

    excluded_timesteps = k_n_functions.find_exclude(inflow)

    inflow_series = inflow["inflow g7"].to_numpy()
    outflow_series = discharge["discharge g7"].to_numpy()

    # select timesteps for error calculation, put False and unselect timesteps that are excluded
    mask = np.array([True] * len(inflow_series))
    mask[excluded_timesteps] = False

    length, slope = load_files.convert(file)
    name = "L{}_S{}".format(length, slope)
    best_error = np.inf
    # create empty table of 0 that has # rows = # n values, # columns = # logks values
    errors = np.zeros((len(s0s), len(slope0s), len(q0s)))
    i = 0
    j = 0
    k = 0
    for s0 in s0s:
        j = 0
        print(round(100 * i / len(s0s), 2))
        for slope0 in slope0s:
            k = 0
            for q0 in q0s:
                SQ_string = conceptual_model_lib.get_SQ_string_general([slope0 / 60], [s0], q0,
                                                                       qmax)
                result = conceptual_model_lib.simulate_pipe_fast(inflow_series, SQ_string, name)
                error = k_n_functions.rms_error(outflow_series[mask], result[mask])
                if error < best_error:
                    best_error = error
                    best_slope0 = slope0
                    best_s0 = s0
                    best_q0 = q0
                errors[i, j, k] = error
                k += 1
            j += 1
        i += 1
    best_SQ_slopes = {
        "file": file,
        "best_slope0": best_slope0,
        "best_s0": best_s0,
        "best_q0": best_q0,
        "slope": slope,
        "length": length
    }

    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    np.save(os.path.join(output_folder, "errors_{}.npy".format(name)), errors)

    return best_SQ_slopes, errors


files_folder = "./data_new"
filenames = os.listdir(files_folder)
# look in the folder where the results are stored to see which files are already done
already_done = os.listdir(folder)
alread_done_processed = [x[7:-4] for x in already_done]
file_list = []
for file in filenames:
    if (file[-4:] == ".csv") and (file[:-4] not in alread_done_processed):
        file_list.append(file)
print("Files left to process:", len(file_list))
```

```
params_list = []
if __name__ == '__main__':
    # find the best valuesfor each file
    for file in file_list:
        print(file)
        params = {
            "file": file,
            "s0s": s0s,
            "q0s": q0s,
            "slope0s": slope0s,
            "output_folder": folder,
            "data_folder": files_folder
        }
        find_best_slope_linear(params)
```

I.6 Simulation of the cities

Code used to define the city network and the various scenarios. Additionally, the code shows the simulation of the scenarios of the two cities and the computations of the measures used.

```
import json
import os
import sys

import clr
import numpy as np
import pandas as pd

import conceptual_model_lib
import k_n_functions
import load_files

sys.path.append(os.path.relpath('./Release'))
clr.AddReference('DtuSmModels')
from DtuSmModels import *

star_city_network = {
    "A1": ["A2"], "A2": ["A3"], "A3": ["A4"], "A4": ["A5"], "A5": ["B5"], "H1": ["A5"],
    "B1": ["B2"], "B2": ["B3"], "B3": ["B4"], "B4": ["B5"], "B5": ["C5"],
    "C1": ["C2"], "C2": ["C3"], "C3": ["C4"], "C4": ["C5"], "C5": ["D6"],
    "D1": ["D2"], "D2": ["D3"], "D3": ["E6"], "D4": ["D5"], "D5": ["D6"], "D6": ["E9"],
    "E1": ["E2"], "E2": ["E3"], "E3": ["F3"], "E4": ["E5"],
    "E5": ["E6"], "E6": ["F6"], "E7": ["E8"], "E8": ["E9"],
    "E9": ["F9"], "E10": ["E11"], "E11": ["F11"],
    "F1": ["F2"], "F2": ["F3"], "F3": ["G3"], "F4": ["F5"], "F5": ["F6"], "F6": ["G5"],
    "F7": ["F8"], "F8": ["F9"], "F9": ["G7"], "F10": ["F11"], "F11": ["G9"],
    "G1": ["G2"], "G2": ["G3"], "G3": ["G4"], "G4": ["G5"], "G5": ["G6"], "G6": ["G7"],
    "G7": ["G8"], "G8": ["G9"], "G9": ["outlet"], "outlet": []
}

strip_city_network = {
    "I1": ["I2"], "I2": ["I3"], "I3": ["K3"], "I4": ["I5"], "I5": ["I6"], "I6": ["K6"],
    "I7": ["I8"], "I8": ["I9"], "I9": ["K9"], "I10": ["I11"], "I11": ["I12"],
    "I12": ["K12"], "I13": ["I14"], "I14": ["I15"], "I15": ["K15"], "I16": ["I17"],
    "I17": ["I18"], "I18": ["K18"],
    "K1": ["K2"], "K2": ["K3"], "K3": ["L3"], "K4": ["K5"], "K5": ["K6"], "K6": ["L5"],
    "K7": ["K8"], "K8": ["K9"], "K9": ["L7"], "K10": ["K11"], "K11": ["K12"],
    "K12": ["L9"], "K13": ["K14"], "K14": ["K15"], "K15": ["L11"], "K16": ["K17"],
    "K17": ["K18"], "K18": ["L13"],
    "L1": ["L2"], "L2": ["L3"], "L3": ["L4"], "L4": ["L5"], "L5": ["L6"], "L6": ["L7"],
    "L7": ["L8"], "L8": ["L9"], "L9": ["L10"], "L10": ["L11"], "L11": ["L12"],
    "L12": ["L13"], "L13": ["outlet"], "outlet": []

}

star_city_comp_coarse = {
    "nodes_in_compartment": {
```

```
        "full_city": list(star_city_network.keys()),
    },
    "comp_network": [
        ("full_city", "outlet")
    ]
}

star_city_comp_medium = {
    "nodes_in_compartment": {
        "compartment2": ["A1", "A2", "A3", "A4", "A5", "H1",
                         "B1", "B2", "B3", "B4", "B5",
                         "C1", "C2", "C3", "C4", "C5",
                         "D4", "D5", "D6",
                         "E7", "E8", "E9",
                         "F7", "F8", "F9", "G7"],
        "compartment1": ["D1", "D2", "D3",
                         "E1", "E2", "E3", "E4", "E5", "E6",
                         "F1", "F2", "F3", "F4", "F5", "F6",
                         "G1", "G2", "G3", "G4", "G5", "G6"],
        "compartment3": ["E10", "E11",
                         "F10", "F11", "G9"],
        "transport_stretch": ["G6", "G7", "G8", "G9", "outlet"],
    },
    "comp_network": [
        ("compartment2", "transport_stretch"),
        ("compartment1", "transport_stretch"),
        ("compartment3", "outlet"),
        ("transport_stretch", "outlet")
    ]
}

star_city_comp_fine = {
    "nodes_in_compartment": {
        "compartment1": ["A1", "A2", "A3", "A4", "A5", "H1", "B5"],
        "compartment2": ["B1", "B2", "B3", "B4", "B5", "C5"],
        "compartment3": ["C1", "C2", "C3", "C4", "C5", "D6"],
        "compartment6": ["D4", "D5", "D6",
                         "E7", "E8", "E9",
                         "F7", "F8", "F9",
                         "G6", "G7", "G8"],
        "compartment5": ["D1", "D2", "D3",
                         "E4", "E5", "E6",
                         "F4", "F5", "F6",
                         "G4", "G5", "G6"],
        "compartment4": ["E1", "E2", "E3",
                         "F1", "F2", "F3",
                         "G1", "G2", "G3", "G4"],
        "compartment7": ["E10", "E11",
                         "F10", "F11",
                         "G8", "G9", "outlet"],
    },
    "comp_network": [
        ("compartment1", "compartment2"),
        ("compartment2", "compartment3"),
        ("compartment3", "compartment6"),
        ("compartment4", "compartment5"),
        ("compartment5", "compartment6"),
        ("compartment6", "compartment7"),
        ("compartment7", "outlet")
    ]
}
star_city_comp_full_network = {
    "nodes_in_compartment": {

    },
    "comp_network": [
    ]
}
for start, end in star_city_network.items():
    if len(end) > 0:
        star_city_comp_full_network["nodes_in_compartment"][start] = [start, end[0]]
        star_city_comp_full_network["comp_network"].append((start, end[0]))
```

```
strip_city_comp_coarse = {
    "nodes_in_compartment": {
        "full_city": list(strip_city_network.keys())
    },
    "comp_network": [
        ("full_city", "outlet")
    ]

}
strip_city_comp_medium = {
    "nodes_in_compartment": {
        "compartment1": ["I1", "I2", "I3", "I4", "I5", "I6", "I7", "I8", "I9",
                         "K1", "K2", "K3", "K4", "K5", "K6", "K7", "K8", "K9",
                         "L1", "L2", "L3", "L4", "L5", "L6", "L7", "L8"],
        "compartment2": ["I10", "I11", "I12", "I13", "I14", "I15", "I16", "I17", "I18",
                         "K10", "K11", "K12", "K13", "K14", "K15", "K16", "K17", "K18",
                         "L8", "L9", "L10", "L10", "L11", "L12", "L13", "outlet"],

    },
    "comp_network": [
        ("compartment1", "compartment2"),
        ("compartment2", "outlet")
    ]

}
strip_city_comp_fine = {
    "nodes_in_compartment": {
        "compartment1": ["I1", "I2", "I3",
                         "K1", "K2", "K3",
                         "L1", "L2", "L3", "L4"],
        "compartment2": ["I4", "I5", "I6",
                         "K4", "K5", "K6",
                         "L4", "L5", "L6"],
        "compartment3": ["I7", "I8", "I9",
                         "K7", "K8", "K9",
                         "L6", "L7", "L8"],
        "compartment4": ["I10", "I11", "I12",
                         "K10", "K11", "K12",
                         "L8", "L9", "L10"],
        "compartment5": ["I13", "I14", "I15",
                         "K13", "K14", "K15",
                         "L10", "L11", "L12"],
        "compartment6": ["I16", "I17", "I18",
                         "K16", "K17", "K18",
                         "L12", "L13", "outlet"],
    },
    "comp_network": [
        ("compartment1", "compartment2"),
        ("compartment2", "compartment3"),
        ("compartment3", "compartment4"),
        ("compartment4", "compartment5"),
        ("compartment5", "compartment6"),
        ("compartment6", "outlet")
    ]

}

strip_city_comp_full_network = {
    "nodes_in_compartment": {

    },
    "comp_network": [
    ]
}
for start, end in strip_city_network.items():
    if len(end) > 0:
        strip_city_comp_full_network["nodes_in_compartment"][start] = [start, end[0]]
        strip_city_comp_full_network["comp_network"].append((start, end[0]))

cities_simulation_folder = "cities_simulations"
with open(os.path.join(cities_simulation_folder, "star_city_network.json"), "w+") as f:
```

```
    json.dump(star_city_network, f)

with open(os.path.join(cities_simulation_folder, "strip_city_network.json"), "w+") as f:
    json.dump(strip_city_network, f)

with open(os.path.join(cities_simulation_folder, "star_city_Coarse.json"), "w+") as f:
    json.dump(star_city_comp_coarse, f)

with open(os.path.join(cities_simulation_folder, "star_city_Medium.json"), "w+") as f:
    json.dump(star_city_comp_medium, f)

with open(os.path.join(cities_simulation_folder, "star_city_Fine.json"), "w+") as f:
    json.dump(star_city_comp_fine, f)

with open(os.path.join(cities_simulation_folder, "star_city_Full_Network.json"), "w+") as f:
    json.dump(star_city_comp_full_network, f)

with open(os.path.join(cities_simulation_folder, "strip_city_Coarse.json"), "w+") as f:
    json.dump(strip_city_comp_coarse, f)

with open(os.path.join(cities_simulation_folder, "strip_city_Medium.json"), "w+") as f:
    json.dump(strip_city_comp_medium, f)

with open(os.path.join(cities_simulation_folder, "strip_city_Fine.json"), "w+") as f:
    json.dump(strip_city_comp_fine, f)
with open(os.path.join(cities_simulation_folder, "strip_city_Full_Network.json"), "w+") as f:
    json.dump(strip_city_comp_full_network, f)

# load parameters for the SQ slopes
SQ_slopes_df = pd.read_csv("results_slopes_big_k.csv").set_index(["slope", "length"], drop=False)[
    ["SQ_slope", "slope", "length"]]

# load parameters for the piecewise SQ function
SQ_slopes_piecewise_df = pd.read_csv("SQ_slope_storage_stepwise.csv")

cities_simulation_folder = "cities_simulations"

city_name = "star_city"
scenario = "Fine"

folder_name = "{}_data".format(city_name)
city_info_df = pd.read_csv(os.path.join(folder_name, "{}_info.csv".format(city_name)))
f = open(os.path.join(cities_simulation_folder, "{}_{}.json".format(city_name, scenario)), "r")
selected_scenario = json.load(f)
f.close()

f = open(os.path.join(cities_simulation_folder, "{}_network.json".format(city_name)), "r")
city_network = json.load(f)
f.close()


def simulate_one(city_name, scenario_name, length_mode, diameter_mode, SQ_slopes_df, city_info_df,
                 use_piecewise_SQ=False, scale_SQ_slope=1):
    """
    Simulates one city configuration based on imput parameters.
    Params:
        - city_name: name of the city
        - scenario_name: name of the scenario
        - length_mode: length mode to use, longest sum (lsum) or weighted average (wavg)
        - diameter_mode: maximum diameter (max) or weighted average (wavg)
        - SQ_slopes_df: the dataframe with info on SQ slope params
        - city_info_df: the dataframe with info from the rational method
        - use_piecewise_SQ: whether to use the piecewise SQ function
        - scale_SQ_slope: scale the slope by a given factor
    """
    folder_name = "{}_data".format(city_name)
    city_info_df = pd.read_csv(os.path.join(folder_name, "{}_info.csv".format(city_name)))
    f = open(os.path.join(cities_simulation_folder, "{}_{}.json".format(city_name, scenario_name)),
             "r")
    selected_scenario = json.load(f)
    f.close()
```

```
    f = open(os.path.join(cities_simulation_folder, "{}_network.json".format(city_name)), "r")
    city_network = json.load(f)
    f.close()

    prm_file_path, info = conceptual_model_lib.create_PRM_file_city(city_network,
                                                                    selected_scenario[
                                                                        "comp_network"],
                                                                    selected_scenario[
                                                                        "nodes_in_compartment"],
                                                                    SQ_slopes_df,
                                                                    city_info_df,
                                                                    SQ_slope_mode="model",
                                                                    city_name=city_name,
                                                                    length_mode=length_mode,
                                                                    slope_mode="wavg",
                                                                    diameter_mode=diameter_mode,
                                                                    use_piecewise_SQ=use_piecewise_SQ,
                                                                    scale_SQ_slope=scale_SQ_slope)

    outputs, _, _ = conceptual_model_lib.run_model(prm_file_path,
                                                   city_network,
                                                   selected_scenario["comp_network"],
                                                   selected_scenario["nodes_in_compartment"],
                                                   city_info_df,
                                                   runoff_folder="{}_data".format(city_name),
                                                   runoff_prefix="{}_catchment_runoff_".format(
                                                       city_name))

    # find the name of the last connection in the network
    for connection, values in outputs.items():
        last_conn = connection

    discharge_dtusm = outputs[last_conn]
    discharge_dtusm[-1] = 0
    discharge_muplus = load_files.read_flow_series(
        os.path.join(folder_name, "{}_discharge.csv".format(city_name)))

    return {
        "rmse": k_n_functions.rms_error(np.array(discharge_dtusm),
                                        discharge_muplus["Discharge outlet"].to_numpy()),
        "nse": conceptual_model_lib.nse(np.array(discharge_dtusm),
                                        discharge_muplus["Discharge outlet"].to_numpy()),
        "discharge_dtusm": discharge_dtusm,
        "discharge_muplus": discharge_muplus,
        "city_name": city_name,
        "scenario_name": scenario_name,
        "diameter_mode": diameter_mode,
        "length_mode": length_mode,
        "info": info
    }


scenarios = {
    "star_city": ["Coarse", "Medium", "Fine", "Full_Network"],
    "strip_city": ["Coarse", "Medium", "Fine", "Full_Network"]
}

length_modes = ["lsum", "wavg"]
diameter_modes = ["max", "avg"]

results = []
# simulate all scenarios
for city_name, scenario_list in scenarios.items():
    for scenario_name in scenario_list:
        for length_mode in length_modes:
            for diameter_mode in diameter_modes:
                print(city_name, scenario_name, length_mode, diameter_mode)
                results.append(simulate_one(city_name, scenario_name, SQ_slopes_df, city_info_df,
                                            use_piecewise_SQ=False))
                print(results[-1]["nse"])
```

```
def find_rain_events(timeseries):
    """
    Finds rain events in timeseries
        -  timeseries
    """
    # to count number of cosecutive 0's
    counter = 0
    # tolerance to allow for some slack
    tolerance = 0.061

    potential_exclude = []
    total_to_exclude = []

    # number of timesteps that are 0
    consecutive_zeros = 60

    consecutive_rain = 20
    rain_counter = 0

    recording_series = True
    potential_rain = []
    all_rains = []
    # timestep is index
    for timestep, value in enumerate(
            timeseries):  # enumerate gives a list made of index of element and the element as output
        # count consecutive 0, append timestep to potential_exclude
        if value < tolerance:
            counter += 1
            potential_exclude.append(timestep)
            if (rain_counter >= consecutive_rain):
                if (sum(timeseries[potential_rain]) > 100):
                    all_rains.append(potential_rain)
            rain_counter = 0
            potential_rain = []
        else:
            rain_counter += 1
            potential_rain.append(timestep)
            if counter >= consecutive_zeros:
                # print(potential_exclude)
                total_to_exclude.append(np.array(potential_exclude))

            potential_exclude = []
            counter = 0

    total_to_exclude.append(potential_exclude)
    total_to_exclude = np.concatenate(total_to_exclude)

    return total_to_exclude, all_rains


def nse(dpm, sm):
    avg_dpm = np.mean(dpm)
    return 1 - np.sum(np.power(dpm - sm, 2)) / np.sum(np.power(dpm - avg_dpm, 2))


def pep(dpm, sm):
    return 100 * (np.max(dpm) - np.max(sm)) / (np.max(dpm))


def pdiff(dpm, sm):
    return np.max(dpm) - np.max(sm)


def ptdiff(dpm, sm):
    return np.argmax(dpm) - np.argmax(sm)


scenarios = {
    "star_city": ["Coarse", "Medium", "Fine", "Full_Network"],
    "strip_city": ["Coarse", "Medium", "Fine", "Full_Network"]
}
```

```
def get_measures(discharge_dpm, discharge_sm, outliers_safety=10):
    """
    Find the measures for all the rain events in the CM discharge.
        - discharge_dpm: the discharge from MU+
        - discharge_sm: the discharge from the conceptual model
        - outliers_safety:
    :return:
    """
    _, all_rains_timesteps = find_rain_events(discharge_dpm)
    pep_results = []
    pdiff_results = []
    ptdiff_results = []
    for rain_timesteps in all_rains_timesteps:
        if len(rain_timesteps) > 60:
            dpm = discharge_dpm[rain_timesteps] / 60
            sm = discharge_sm[rain_timesteps] / 60

            if (ptdiff(dpm, sm) < -outliers_safety) or (ptdiff(dpm, sm) > outliers_safety):
                ptdiff_results.append(None)
                pep_results.append(None)
                pdiff_results.append(None)

            else:
                ptdiff_results.append(ptdiff(dpm, sm))
                pep_results.append(pep(dpm, sm))
                pdiff_results.append(pdiff(dpm, sm))
    return pep_results, pdiff_results, ptdiff_results
```